Copy 47 of 60 copies

2

IDA PAPER P-2259

AN AGGREGATOR PREPROCESSOR FOR NAVMOD

Eleanor L. Schwartz

June 1990

Prepared for
Joint Chiefs of Staff

IDA
INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311-1772

IDA Log No. HQ 89-34634

DEFINITIONS

IDA publishes the following documents to report the results of its work.

Reports

Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

Group Reports

Group Reports record the findings and results of IDA established working groups and panels composed of senior individuals addressing major issues which otherwise would be the subject of an IDA Report. IDA Group Reports are reviewed by the senior individuals responsible for the project and others as selected by IDA to ensure their high quality and relevance to the problems studied, and are released by the President of IDA.

Papers

Papers, also authoritative and carefully considered products of IDA, address studies that are narrower in scope than those covered in Reports. IDA Papers are reviewed to ensure that they meet the high standards expected of refereed papers in professional journals or formal Agency reports.

Documents

IDA Documents are used for the convenience of the sponsors or the analysts (a) to record substantive work done in quick reaction studies, (b) to record the proceedings of conferences and meetings, (c) to make available preliminary and tentative results of analyses, (d) to record data developed in the course of an investigation, or (e) to forward information that is essentially unanalyzed and unevaluated. The review of IDA Documents is suited to their content and intended use.

The work reported in this document was conducted under contract MDA 903 89 C 0003 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that Agency.

This Paper has been reviewed by IDA to assure that it meets high standards of thoroughness, objectivity, and appropriate analytical methodology and that the results, conclusions and recommendations are properly supported by the material presented.

This Paper does not necessarily represent the views of the Joint Chiefs of Staff for whom it was prepared and to whom it is forwarded as independent advice and opinion.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1990	

4. TITLE AND SUBTITLE
An Aggregator Preprocessor for NAVMOD

5. FUNDING NUMBERS
C-MDA 903 89 C 0003
TA-T-I6-682

6. AUTHOR(S)
Eleanor L. Schwartz

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria, VA 22311

8. PERFORMING ORGANIZATION REPORT NUMBER
IDA Paper P-2259

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Joint Staff, J-8
The Pentagon
Washington, D.C. 20301

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

Combat

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This paper documents an "aggregator preprocessor" computer program that has been developed to facilitate certain aspects of data generation for the NAVMOD naval model. NAVMOD simulates only a limited number of types of different naval resources; input data for NAVMOD must be expressed in terms of these types. The aggregator preprocessor reads data for actual types of naval weapons or other resources. Using a weighted averaging technique, the computer program computes aggregated data values suitable for use by NAVMOD and outputs them in a format that NAVMOD can read. The weights used in the averaging procedure are based on user inputs. This allows the user to specify the relative compositions of the NAVMOD resource types in terms of actual types of naval resources. The program develops aggregated resource effectiveness values that are consistent with these relative compositions. The program makes extensive use of the INGRES database management system; it is written in FORTRAN with embedded INGRES database commands. It is designed to run on a Digital Equipment Corporation VAX computer with the VMS operating system, but it probably can be converted to run on some other machines.

14. SUBJECT TERMS
Naval combat modeling, campaign level combat, NAVMOD Naval Model, user interfaces, computer simulation, data preprocessors, preprocessors, database applications, database management, INGRES database

15. NUMBER OF PAGES
268

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

IDA PAPER P-2259

AN AGGREGATOR PREPROCESSOR FOR NAVMOD

Eleanor L. Schwartz

June 1990

INSTITUTE FOR DEFENSE ANALYSES

Contract MDA 903 89 C 0003
Task T-16-682

PREFACE

The Institute for Defense Analyses has prepared this paper under Contract No. MDA 903 89C 0003, Task Order T-I6-682, Net Assessment Methodologies and Critical Data Elements for Strategic and Theater Force Comparisons, for the Joint Staff, Director for Force Structure, Resource and Assessment, J-8. The paper documents an aggregator preprocessor computer program for the NAVMOD model of naval combat. The author is grateful to Dr. Lowell Bruce Anderson and Mr. John Cook, of IDA, for their review of this paper.

ABSTRACT

This paper documents an "aggregator preprocessor" computer program that has been developed to facilitate certain aspects of data generation for the NAVMOD naval model. NAVMOD simulates only a limited number of types of different naval resources; input data for NAVMOD must be expressed in terms of these types. The aggregator preprocessor reads data for actual types of naval weapons or other resources. Using a weighted averaging technique, the program computes aggregated data values suitable for use by NAVMOD and generates output of these values in a format that NAVMOD can read. The weights used in the averaging procedure are based on user inputs. This allows the user to specify the relative compositions of the NAVMOD resource types in terms of actual types of naval resources. The program develops aggregated resource effectiveness values that are consistent with these relative compositions. The program makes extensive use of the INGRES database management system; it is written in FORTRAN with embedded INGRES database commands. It is designed to run on a Digital Equipment Corporation VAX computer with the VMS operating system, but it probably can be converted to run on some other machines.

NOTE

The NAVMOD preprocessors were developed in INGRES Release 5. INGRES Release 6 is now on the market. Some minor reprogramming of the preprocessors might be necessary to have them work under INGRES Release 6.

CONTENTS

APPENDICES

FIGURE

TABLES

I. OVERVIEW OF THE AGGREGATOR PREPROCESSOR

A. A PREPROCESSOR FOR NAVMOD

1. Introduction

This paper documents an "aggregator preprocessor" computer program that has been developed to facilitate certain aspects of generation of input data for the NAVMOD naval model. NAVMOD is an aggregated, deterministic model of naval combat. It is documented in References [1] and [2]. A FORTRAN computer program, NAVMOD accepts values for several hundred input variables and arrays, encompassing several thousand pieces of numerical information. As discussed in Appendix A of Reference [1] and Chapter III, Section D, of Reference [2], these input data must be arranged in a file in a certain specified format. The aggregator preprocessor program accepts and operates on certain sets of data and generates output files that can then be used (read) by NAVMOD itself. A second preprocessor for NAVMOD, the "interaction selector," has also been developed, and is documented under separate cover.

The aggregator preprocessor was developed to assist in the preparation of realistic data values for the NAVMOD inputs. Available data values are, in general, a function of resource type. There are a number of resources, however, of which NAVOD can simulate only one type. For other resources, NAVMOD can simulate a user-input number of "notional" types, but computer time and/or storage space usage might become prohibitively large if more than relatively few types are played (see Reference [1], Chapter II, Section B.4 and Reference [2], Chapter III, Sections A and B). The aggregator preprocessor allows the user to specify the relative composition of the notional (NAVMOD) resource types in terms of actual resource types, and then computes appropriate weighted averages of a set of "more disaggregated" data, which are (in general) stated in terms of actual resource types. The resulting "aggregated data" are output in such a format that they can then be used as input to NAVMOD. That is, from the more disaggregated data, the

aggregator preprocessor constructs a set of NAVMOD input data.[1] The preprocessor automates the numerous hand computations that would be required to average the data for the actual resource types.

In running the aggregator preprocessor, the user specifies the particular actual resource types to be considered. Once the aggregator has been run, many NAVMOD runs can be performed with the aggregated data, varying the interactions selected and/or number of notional resources played--with the understanding that these notional resources represent the particular weighted mixes of actual resource types that were considered in the run of the aggregator. When the user wants to consider different mixes of actual resource types, the aggregator preprocessor can be run again.[2]

The aggregator preprocessor is designed for the NAVMOD model as documented in References [1] and [2]. If the input variables to NAVMOD are changed, it is possible under some circumstances to modify the preprocessor to work with the changed version of NAVMOD. Some procedures, including some auxiliary computer programs, have been developed to aid in the modification; these procedures are discussed in Chapter IV, Section C. An attempt has been made to keep the algorithms of the aggregator as generalizable as possible and to put NAVMOD-specific constructs in the database rather than the computer code. The aggregator methodology might be able to be adapted fairly easily to other combat models, without a large amount of reprogramming.

2. Motivation--Why an Aggregator?

As mentioned in the preceding section, the aggregator preprocessor was developed to facilitate converting more realistic data, which tend to be defined for specific actual resource or weapon types, into data usable by the NAVMOD model. An alternative way of handling actual resource types would be to alter NAVMOD so that it could consider an arbitrary number of types of each resource it models and to then use the more disaggregated data directly (letting the resource types modeled by NAVMOD correspond one-for-one to the actual resource types the user desires to consider). One problem with this approach is the difficulty in recoding NAVMOD to handle multiple weapon types. There are some

---

[1] Note that both the more disaggregated data and the resultant aggregated data are for initial values only, not for mid-run changes to be applied by Subroutine TIMET of NAVMOD.

[2] In this context, a "mix" of resource types can in fact consist of a single resource type. Thus, the aggregator can facilitate the examination of the effects of substituting one type of resource for another--for example, all A-6s as opposed to all A-7s on ground attack missions.

resources of which NAVMOD models only one type. Modifying the NAVMOD code to consider multiple numbers of types of these resources would be possible, but tedious and time-consuming to implement.

The other problem with modeling a large number of types of each resource is that the computer running time (and for some machines, storage requirements) might increase considerably as the number of types of resources modeled increases. In NAVMOD, there might be 20 to 30 actual types of Blue ships for which data are available; there might also be fairly large numbers of actual types of other resources. The use of a smaller number of notional resource types could keep the running time of the model from becoming prohibitively long, and might allow the model to be run on smaller computers. Also, analysis of the output of the model might be easier if relatively fewer types of resources are displayed.

As noted previously, the aggregator generates effectiveness data that are consistent with notional resources that represent certain mixes of certain actual resource types. If one particular set of mixes does not adequately capture the scope of the problem to be analyzed, the user can examine the effects of a different set of mixes by changing the resource and allocation fraction variable values to reflect the new mixes (see Chapters II and III), running the aggregator again, and then running the model with the newly generated data.

Although the aggregator preprocessor requires a relatively extensive run time, it is separate from NAVMOD, and many runs of NAVMOD (which runs quite quickly) can be made for each run of the aggregator preprocessor. Separation of the aggregator from NAVMOD itself allows the two programs to be run on different machines. For instance, the aggregator can be run on a large, fast computer with INGRES installed on it. Its output data files (which are input to NAVMOD) can be then transferred to a smaller machine (such as a PC) on which NAVMOD is run.

Even within the aggregator, some grouping of actual resource types might be necessary. The greater the number of resource types considered, the longer the running time of the aggregator program.

3. A Precis of INGRES

The aggregator preprocessor program uses the INGRES database management system: The program is written in FORTRAN with embedded SQL commands. Although knowledge of INGRES is not necessary to use the programs, some familiarity with

INGRES will facilitate understanding the programs and possible later expansion of them. (See References [3] and [4] for a more extensive overview of INGRES and its capabilities.)

The preprocessors use the INGRES version that is designed for a Digital Equipment Corporation VAX computer running the VMS operating system. Versions of INGRES exist for other computers, including the IBM PC, and converting the preprocessor programs to run on such computers may not be difficult. However, some changes to the programs would probably be required. (NAVMOD itself has been run on an IBM PC.)

a. Databases and Forms

INGRES (originally an abbreviation of Interactive Graphics and Retrieval System) is a relational database management system. It operates on a database, which in turn is a collection of tables. INGRES allows the user to create tables as desired. Each table is a rectangular array of information and must be given a distinct name. The number of columns in the array is fixed when the table is created, as is the type (integer, character, or floating point) and length (in bytes) of each column. Each column is also named. The number of rows in the array can vary, and adding rows to a table and/or deleting rows from a table is possible. Each cell (row/column combination) in a table contains a piece of information of the type and length appropriate to the cell's column. The preprocessor program documented here uses a database named NAVPRE, with 32 tables. The specification and contents of NAVPRE should be considered as being an essential part of the preprocessor.

(The aggregator preprocessor and the interaction selector preprocessor both use the same INGRES database. With this exception, the two programs are completely independent of each other. The user of NAVMOD can run either or both programs, as suitable. In fact, only 2 of the 32 tables in the database are used by both programs, and if desired, creating a separate database for each preprocessor would not be difficult.)

INGRES also has a capability to use forms as an interface for data transfer. An INGRES form looks much like an ordinary paper form. Various parts of INGRES allow the user to create forms and display them on a terminal, allow the forms to display certain data, and allow the user to enter certain information into them from a terminal. The interaction selector preprocessor program uses 17 different forms, which can be provided

along with the computer program. The aggregator preprocessor, however, does not use any forms.

INGRES supplies a number of ways to retrieve, present, manipulate, and/or change data in a database. These ways can be roughly grouped into interactive query languages, embedded query languages, and forms-based INGRES subsystems; these methods are discussed in the following paragraphs. (Note that knowledge of INGRES is not necessary to run the preprocessor program.)

b. Interactive Query Languages

INGRES supplies two interactive query languages, named QUEL (mnemonic for query language) and SQL (structured query language). Each language consists of a set of commands that can be entered at a terminal. Although the two languages differ considerably in detail, they perform essentially the same types of operations. (Either language can be invoked on a database.) Some of the commands retrieve information from one or more tables in a database--the retrieved information is displayed on the terminal. Commands exist to retrieve from a specified table only those rows that contain information satisfying a specified search condition. Other commands can be used to change the information in a table, add rows to a table, and delete rows from a table. Additionally, the contents of a table can be written to an ordinary ASCII file, and an ASCII file can be read into an INGRES table. The aggregator preprocessor utilizes the latter procedure.

The query languages can aid in user update of the database, should this become desirable to expand the scope of the program (as discussed in Chapter IV). SQL is documented in References [5] and [6] and QUEL in References [7] and [8].

c. Embedded Query Languages

INGRES has two embedded query languages, Embedded SQL and EQUEL (Embedded QUEL). The existence of the embedded query languages is the main reason INGRES was used for the preprocessors. These languages include sets of commands that can be embedded in--invoked from--a computer program written in a language such as FORTRAN. (The discussion here relates to FORTRAN, but INGRES also supports embedded commands in PASCAL, C, Ada, and several other languages.) The discussion here focuses on Embedded SQL, which is documented in References [9] and [10]. EQUEL is similar in spirit to Embedded SQL, although considerably different in detail.

Embedded SQL commands can be categorized as embedded database commands or embedded forms commands. Embedded database commands are analogs of interactive SQL commands. They can do things such as retrieve information from a database and assign the retrieved values to certain program (FORTRAN) variables (which can then be manipulated by the program). The current value of some program variable can be used as part of a search condition. There are also embedded database commands that can perform the following:

- Set certain entries in a database to the current value of some program variable
- Insert rows into a database table
- Delete rows from a table
- Copy an ASCII file into a table, or vice versa.

Embedded forms commands operate on forms that have been created (earlier) via the INGRES forms editor (Reference [11]). Although the aggregator does not use embedded forms commands, a few words about them are in order. These commands are designed for use in interactively-run programs. Embedded forms commands can do such things as

- Display a form on the terminal
- Display the values of certain program variables in a form
- Allows the user to type information into (certain parts of) a form
- Read information from a form and have this information be assigned to a program variable(s)
- Display a user-written help file.

Embedded forms commands also allow a user to create frames and menu items. For the purposes of this discussion, a menu item can be regarded as a portion of code (consisting of a mixture of FORTRAN statements, embedded database commands, and embedded forms commands) that can be invoked from a terminal during program execution. A frame is a combination of a form and two or more menu items in which the form is displayed on the terminal, and the user can choose which menu item to invoke (by pressing certain keys). There are embedded forms commands to display frames and create menu items. Certain embedded forms commands allow the execution of a menu item to result in a new frame's being displayed.

The use of embedded query languages clearly expands the methods by which data can be entered into a FORTRAN program. In a regular FORTRAN program, data can be read from a flat (ASCII) file and data can be typed into a terminal and read by a FORTRAN "READ" statement. With embedded database commands, it is also possible to retrieve data values from an INGRES database and assign them to program variables, and read data from a flat file into an INGRES data table, where they can then be operated on by INGRES commands or accessed from the INGRES data table. With embedded forms commands, data can be typed into a form, read by a forms command, and assigned to a program variable. In a single program, all of these methods can be used, in any combination, to yield maximum flexibility in the handling of input.

The aggregator preprocessor is written in FORTRAN with embedded INGRES database commands but not with embedded INGRES forms commands and need not be run interactively. Since the aggregator preprocessor requires extensive run time, the user might prefer to run it as a batch job. The preprocessor uses a variety of INGRES tables and ASCII files, which will be explained in paragraphs that follow.

d. INGRES Forms-Based Subsystems

INGRES includes several forms-based subsystems--programs that can perform various operations associated with a database. These programs use frames and menus and operate much like a program written in embedded query language with embedded forms commands, but these programs are supplied by INGRES, rather than written by the user. For the purposes of this paper, the most important INGRES forms-based subsystem is QBF (Query-by-Forms). QBF can perform most of the functions of the interactive query languages, but within a frame-based context that obviates the need for the user to learn query language commands. With QBF, the user can retrieve or change data in a database merely by typing information into certain forms. QBF is documented in Reference [12].

Additional INGRES forms-based subsystems include a report writer, a forms editor (which allows one to create and edit forms), a graphics capability, and ABF (Applications-by-Forms), a subsystem that allows one to create sequences of frames and menu items by a method that itself uses forms and frames. (The end results of ABF are quite similar in operation to embedded SQL programs that use embedded forms commands.) In addition, INGRES/MENU, a forms-based subsystem, allows the other subsystems and the two interactive query languages to be accessed through menus.

Embedded database commands do exist that allow the user to access the forms-based subsystems from within a program written in embedded query language; however, neither of the two preprocessors use such commands.

Reference [4] provides a fairly extensive discussion of the INGRES forms-based subsystems. Reference [13] documents INGRES/MENU and provides references to the documentation of the other forms-based subsystems. Reference [14] provides some general information on using a terminal with INGRES forms.

The preprocessor programs operate essentially independently of the forms-based subsystems, and the knowledge of the forms-based subsystems is not necessary to use the preprocessor programs. If it becomes desirable to change the contents of the database in order to expand the scope of the preprocessor programs, QBF (which is quite easy to learn) might be useful. (Note: The user is cautioned that the database contents should be changed only in accordance with the procedures outlined in this paper. Errors can occur if the database is updated indiscriminately.)

4. Organization of Paper

This paper is structured as follows. The remaining part of Chapter I constitutes an overview of the aggregator preprocessor. It provides illustrative examples of the more disaggregated data, and introduces the concepts underlying the program and the INGRES database. This chapter concludes with a discussion of the mechanics of running the computer program.

Chapter II describes the basic concepts and algorithms underlying the preprocessor program and the INGRES database, and discusses, in some detail, the data requirements. Chapter II also contains descriptions of many of the INGRES tables used. Chapter III describes the algorithms used to aggregate the data; it includes discussions of each subroutine of the computer program.

Chapter IV discusses a number of specific items of which the user should be aware, including database ownership issues and certain input variables that might need special treatment. Chapter IV also addresses the problem of setting up or altering the information in the INGRES database. The database can be delivered with sufficient information to run the program. There is some amount of leeway in the contents of the database, however, and the user can tailor certain portions of the database as desired. A series of procedures involving auxiliary files and computer programs has been developed to implement this

tailoring so that the database is updated properly. These procedures, and the auxiliary files and programs, are discussed in Chapter IV.

Appendix A contains tables that provide certain detailed information about the computer program, and Appendix B describes the relevant tables in the INGRES database. Appendix C provides an index of NAVMOD and aggregator input variables.

B. OVERVIEW OF CONCEPTS

This section provides an overview of the concepts that underlie the algorithms and database structure of the aggregator preprocessor. Section B.1 contains examples of the form of the more disaggregated data. Sections B.2 and B.3 provide a brief introduction to the material of Chapters II and III.

1. The More Disaggregated Data--An Overview

As stated earlier, the aggregator preprocessor takes a set of more disaggregated data and computes from it data suitable for input to NAVMOD. Values for the more disaggregated data must be supplied by the user of the preprocessor, and thus knowledge of the structure and form of the more disaggregated data is necessary to use the preprocessor. This section illustrates this structure through some examples. The next section explains the form of the data by detailing some constructs associated with it in the preprocessor.

a. Actual Resources for NAVMOD Dimensions

As a first example, consider the NAVMOD variable[3] SSRBES, which is used in the combat interaction between Red and Blue surface ships (Subroutine SHPSHP). Its definition, taken from Appendix F of Reference [2], is

> SSRBES(KBSS)--Number of incoming Red ship-to-ship missiles that can be engaged by the SSD systems of one type-KBSS Blue ship. KBSS=1--carriers; KBSS=2,...,NKBES+1--escort ships, by type; KBSS=NKBES+2--URG ships. This input is not used if ISSLAM=1.

In NAVMOD, this input is a one-dimensional array. The different elements along the dimension correspond to the different types of Blue surface ships that NAVMOD can model: one notional type of carrier, NKBES notional types of escort ships (the limit variable NKBES is itself an input to NAVMOD (and to the aggregator preprocessor)), and

---

[3] In this paper, the term "variable" can refer to both scalars and arrays.

one notional type of URG ship. NAVMOD reads in a data value for each of these array elements. This dimension can be considered as encompassing three different resource categories: (Blue) carriers, escort ships, and URG ships. The concept of resource category plays a key role in the preprocessor. The number of elements along this dimension that will be used by the NAVMOD code is NKBES+2 (i.e., 1+NKBES+1).

In the more disaggregated data, the notional types of ships are replaced by actual types of ships. The more disaggregated data for the variable SSRBES could appear similar to the data displayed in Figure I-1. Each line of data contains the variable name (SSRBES), the name of an actual type of surface ship, and a data value. The data values have the definition:

> SSRBES for an indicated Blue surface ship type equals the number of incoming Red ship-to-ship missiles that can be engaged by the SSD systems of one Blue surface ship of the indicated type.

(The actual values appearing in Figure I-1 and all other numerical values contained in this section are entirely hypothetical and are not intended to be realistic.)

The meaning of the more disaggregated variable is similar to that of the NAVMOD variable, but notional ship type has been replaced by an actual type of ship. The indicated types of Blue surface ships range over all the types of actual Blue carriers, escort ships, and URG ships that the user wants to consider. As part of the setup of the INGRES database (before the aggregator program itself is run), the user can edit a certain INGRES table so that it contains lists of the desired actual resource types. With the current formatting, each actual resource type is identified by a character string that is no more than 10 characters long; this limit could be expanded if desired.[4]

The more disaggregated data are stored on a series of ASCII files, from which they are read into an INGRES table by the program. Auxiliary programs exist that generate shell files, which consist of lines that have the variable name and the names of the appropriate actual resource types, in the correct format, but with zeros for the (more disaggregated) data values. The user can insert the appropriate data values with a text

[4] The specification of the actual resource types can still involve some sort of grouping of resource types. There is a case to be made that each actual platform is different. If this difference is considered significant, each platform can be treated as a different resource type, which might make sense for carriers. Note, however, that the amount of more disaggregated data required can grow rapidly as more actual resource types are specified.

SSRBES CVN-68	3.50000
SSRBES CV-67	4.00000
SSRBES CVN-65	4.50000
SSRBES CV-63	5.00000
SSRBES CV-59	5.50000
SSRBES CV-41	6.00000
SSRBES CG-47	10.00000
SSRBES CG-52	11.00000
SSRBES CG-26	12.00000
SSRBES CG-16	13.00000
SSRBES CGN-38	14.00000
SSRBES CGN-35	15.00000
SSRBES CGN-36	14.50000
SSRBES CGN-25	14.00000
SSRBES CGN-9	13.50000
SSRBES BB-61	13.00000
SSRBES DDG-993	8.50000
SSRBES DDG-51	8.00000
SSRBES DDG-37	7.50000
SSRBES DDG-2	7.00000
SSRBES DD-963	6.00000
SSRBES DD-963 ABL	5.50000
SSRBES DD-963 VLS	5.00000
SSRBES FFG-7	2.00000
SSRBES FFG-1	3.00000
SSRBES FF-1052	4.00000
SSRBES FF-1040	3.00000
SSRBES AOE	2.00000
SSRBES AOR	2.00000
SSRBES AO	1.50000
SSRBES AFS	1.00000

Figure I-1. EXAMPLE OF MORE DISAGGREGATED DATA FOR VARIABLE SSRBES

editor. (These auxiliary programs are described in Chapter IV.) The aggregator constructs values for the (array elements of the) NAVMOD variable SSRBES by taking weighted averages of (certain of the) more disaggregated data values for SSRBES. A different weighted averaging is performed for each of the NKBES+2 elements of the NAVMOD input array. The value for the first element, corresponding to carriers, consists of a weighted average of the more disaggregated values for the actual types of carriers considered. The value for the (NKBES+2)th element, corresponding to URG ships, consists of a weighted average of the more disaggregated values for the actual types of URG ships considered. And for each of the NKBES notional types of escort ships, the aggregator computes a different weighted average of the more disaggregated data values for the actual types of escort ships considered. The weights used in these weighted averages are computed by the preprocessor, based on values of resource variables.

As a second example, consider the NAVMOD input ACPKSC, a two-dimensional array, used in the airbase attack portion of NAVMOD:

> ACPKSC(KRSAM,KBC)--Probability that a Red SAM of type KRSAM kills a Blue cruise missile of type KBC on airbase attack, given that the SAM is shooting at the missile. KRSAM=1,...,NABSAM. KBC=1 for surface-launched cruise missiles, KBC=2 for submarine-launched cruise missiles.

In NAVMOD, the first dimension of this variable corresponds to notional type of Red SAM, of which NAVMOD can play NABSAM different types (the limit variable NABSAM is an input to NAVMOD and to the aggregator), and the second dimension corresponds to notional type of Blue sea-launched land-attack cruise missile, of which NAVMOD plays two notional types, one surface launched and one submarine launched. NAVMOD thus uses NABSAM*2 elements of this array. (The symbol "*" denotes multiplication.) In constructing the more disaggregated data for this variable, the notional types of Red SAMs are replaced by a list of actual types of Red SAMs that might be used for airbase defense, and the notional types of Blue sea-launched land-attack cruise missiles are replaced by a list of actual types of Blue surface-launched and/or submarine-launched cruise missiles. The set of more disaggregated data for ACPKSC might appear as follows:

```
ACPKSC SA-2     TLAM                    0.41000
ACPKSC SA-2     TLAM/C                  0.51000
ACPKSC SA-2     TLAM/D                  0.61000
ACPKSC SA-3     TLAM                    0.45000
ACPKSC SA-3     TLAM/C                  0.55000
ACPKSC SA-3     TLAM/D                  0.65000
```

ACPKSC SA-5	TLAM	0.49000
ACPKSC SA-5	TLAM/C	0.59000
ACPKSC SA-5	TLAM/D	0.69000

Note that a data line exists for each combination of actual Red SAM type and actual Blue cruise missile type. The particular actual resource types used have been pre-specified by the user and stored in the INGRES database. For each of the NABSAM*2 elements of the NAVMOD input array (those elements that are, in fact, used), the aggregator computes a different weighted average of these more disaggregated data values. The weights are computed by a fairly involved, multi-step procedure.

Consider a third example, the variable WRLNDQ, used in the Red air versus Blue ships combat interaction of NAVMOD. Its definition, as used in NAVMOD, is

> WRLNDQ(L)--Range (of the air attack from task force center) that land-based resources are able to warn the task force of an impending Red air attack given that these land resources are able to provide this warning, when the task force is in region L.

This variable is a one-dimensional array, but the different elements along the dimension do not correspond to different types of resources. Instead, they correspond to different regions where the task force might be located; each region is identified by a number. (See Reference [1], Chapter I, Section C, for a description of NAVMOD's modeling of regions.) The aggregator does not attempt to average values over different regions; each region is treated separately. The more disaggregated data values would be exactly the same as the NAVMOD data values, except they would appear in a different format. The more disaggregated data might appear as follows:

WRLNDQ 01	200.00000
WRLNDQ 02	150.00000
WRLNDQ 03	100.00000
WRLNDQ 04	0.00000
WRLNDQ 05	0.00000
WRLNDQ 06	0.00000
WRLNDQ 07	0.00000
WRLNDQ 08	0.00000
WRLNDQ 09	0.00000
WRLNDQ 10	0.00000
WRLNDQ 11	0.00000
WRLNDQ 12	0.00000

For this variable, the number of elements used by the NAVMOD code equals the number of regions played, which is given by the value of the limit variable NLOC, but the size of the array might be larger, given by the value of the symbolic constant MXLOC. The value of NLOC is read by the aggregator before the aggregation algorithm proceeds.

The data shell file would have MXLOC data lines in it (12, in the preceding example), but the aggregator would access only the first NLOC data lines. For each such line, the aggregator would simply copy the data value to the corresponding NAVMOD array element. (More precisely, a weighted average of one more disaggregated data value, with a weight of 1.0, would be performed for each element of the NAVMOD array that would be used by the NAVMOD code.)

Some NAVMOD variables are two- or three-dimensional arrays in which one dimension is region (or some other numerical dimension) and the other dimensions are resource-based dimensions. The more disaggregated data would have a separate data line for each appropriate region-number/actual-resource-type combination. For each combination of region and notional (NAVMOD) resource type, a weighted averaging of the more disaggregated data would be performed, but the averaging would be over only the different actual resource types; the region number would simply be held fixed at the appropriate value.

b. Added Dimensions

In the preceding examples, the number of dimensions associated with the more disaggregated data was the same as the number of dimensions of the NAVMOD variable. The aggregator preprocessor also allows the user to specify added dimensions for a variable, so that the more disaggregated data has more dimensions than the NAVMOD variable. The aggregation algorithm then "aggregates down" these extra dimensions. The preprocessor allows several different types of added dimensions. Within these types, the user can invent specific dimensions and (subject to a few restrictions) can specify which dimensions are to be added to which NAVMOD variables. The use of added dimensions can allow the more disaggregated data used by the preprocessor to be closer to the actual data available. (Note, however, that the amount of more disaggregated data required grows exponentially with the number of added dimensions.)

The aggregator treats added dimensions somewhat differently than the dimensions that appear in the NAVMOD variable; the treatment depends on the type of added dimension. This section presents a few examples of how added dimensions can be used. (More complete explanation appears throughout the rest of the paper.)

One use of an added dimension is to specify actual resource types for a variable that pertains to a resource that NAVMOD models as only one notional type. For example,

consider the variable SUBSOR, used in the Red submarine versus Blue task force combat interaction:

> SUBSOR--The distance from task force center to the torpedo release line (nautical miles). Used in Subroutine CTFMOD.

In NAVMOD, this variable is a scalar. The distance from task force center at which a Red torpedo-firing submarine releases its torpedoes could conceivably depend on the type of submarine. NAVMOD models only one notional type of such submarine, however, and thus having a dimension on submarine type in the NAVMOD variable is not needed. In the more disaggregated data, a dimension on type of Red torpedo-firing submarine could be added to SUBSOR, so that this variable would be zero-dimensional in NAVMOD but one-dimensional in the aggregator preprocessor. A list of actual types of Red torpedo firing submarines could be set by the user and stored in the database. The more disaggregated data for the variable SUBSOR might appear as follows:

```
SUBSOR ALFA                                             7.00000
SUBSOR VICTOR I                                         6.00000
SUBSOR NOVEMBER                                         5.00000
SUBSOR TANGO                                            3.00000
SUBSOR AKULA                                            2.00000
SUBSOR SIERRA                                           3.00000
SUBSOR MIKE                                             4.00000
SUBSOR YANKEE                                           5.00000
SUBSOR VICTOR II                                        6.00000
SUBSOR VICTOR III                                       6.50000
SUBSOR ECHO                                             7.00000
SUBSOR WHISKEY                                          8.00000
SUBSOR KILO                                             9.00000
```

The preprocessor would compute a set of weights, one for each actual submarine type in the list. These weights could be interpreted as the relative proportions of the different actual submarine types that the one notional type of Red torpedo-firing submarine (that NAVMOD can play) is composed of. The preprocessor would then use these weights to average the more disaggregated data values for SUBSOR into one value suitable for the NAVMOD scalar input variable SUBSOR.

Added dimensions can be used to make a variable a function of additional combat-related parameters. For example, a NAVMOD input detection probability that is a function only of type of searcher can be given in the more disaggregated data an added dimension on type of target. A kill probability that in NAVMOD is a function of shooter type and target type can be given in the more disaggregated data an added dimension on munition type.

(NAVMOD--with a few exceptions--models only one notional type of each type of munition it models; see Reference [2], Chapter I.) The added dimension might allow the more disaggregated data to be closer to data from existing sources (which might take into account the added dimension).

The following example indicates both the possibilities and the complexities of using an added index. Consider the NAVMOD input ABPDS, used in the Red airbase defense interaction. Its definition is

> ABPDS(KRSAM)--Probability that a particular Red SAM will detect a particular Blue aircraft. KRSAM=1,...,NABSAM. (Subroutine ABATCK)

This input is indexed by searcher type notional type of Red SAM. The number of such types is given by the limit variable NABSAM. The more disaggregated data substitutes actual types of Red SAMs for the notional types.

Suppose the user wanted to add a dimension for type of target. This variable applies in the combat interaction between Red SAMs and Blue sea-based aircraft on airbase attack. These aircraft encompass attack aircraft plus fighter aircraft assigned to attack missions. NAVMOD models the resource categories Blue sea-based attack aircraft and Blue sea-based fighter aircraft. The user can specify that the combination of these resource categories be used as an added dimension to ABPDS. (The procedure for doing this is explained in Chapter IV, Section D.2.) As indicated earlier, the user specifies the actual resource types in each resource category. The shell file for the more disaggregated data would contain a data line for each combination of actual Red SAM type and actual Blue attack or fighter aircraft type. After data values were edited in, the more disaggregated data may be similar to the following:

```
ABPDS  SA-2    A-6                        0.11000
ABPDS  SA-2    A-7                        0.12000
ABPDS  SA-2    F/A-18                     0.13000
ABPDS  SA-2    S-3                       -1.00000
ABPDS  SA-2    F-14                      -1.00000
ABPDS  SA-3    A-6                        0.21000
ABPDS  SA-3    A-7                        0.22000
ABPDS  SA-3    F/A-18                     0.23000
ABPDS  SA-3    S-3                       -1.00000
ABPDS  SA-3    F-14                      -1.00000
ABPDS  SA-5    A-6                        0.36000
ABPDS  SA-5    A-7                        0.35000
ABPDS  SA-5    F/A-18                     0.34000
ABPDS  SA-5    S-3                       -1.00000
ABPDS  SA-5    F-14                      -1.00000
```

Each data value represents the probability that a Red SAM of the indicated type will detect a Blue aircraft of the indicated type. The use of negative values lets the user mark infeasible resource combinations. A user might not want an F-14/Red-SAM interaction to be modeled because the F-14 would be assigned to escort, not attack, missions and would not get close enough to the SAMs to engage them. Similarly, S-3 aircraft might be considered in some scenarios as being able to perform non-ASW attack missions, and thus could be included in the list of actual resource types for Blue sea-based attack aircraft, but S-3s would probably not be assigned to ground attack. These infeasible resource combinations are not considered in the weighted averaging procedure. (See Chapter III, Section H.1, for more information.)

The preprocessor computes (using resource variable values) a set of weights for the different types of actual Blue sea-based attack and fighter aircraft considered. These weights can be interpreted as representing the relative proportions of the different aircraft types in the target set. For each of the NABSAM notional types of Red SAM, the preprocessor has computed a set of weights that express the notional type as a weighted combination of actual types (these weighted combinations could simply correspond to one actual type per notional type). For each notional type of SAM, the preprocessor computes a (different) weighted average of the more disaggregated data values, using the (products of the) appropriate SAM weights and aircraft weights, but does not include infeasible resource combinations. This computation results in an aggregated data value for the array element of ABPDS that corresponds to that notional type of SAM. (This procedure is described in detail in Chapter III, Section H.)

Specifying two or more added dimensions for a variable is possible. The preprocessor requires the total number of dimensions (NAVMOD plus added) for each variable to be less than or equal to 4. With relatively minor format changes, this limit could be increased to 6, and with relatively minor programming changes, it could be further increased. Note that most of the NAVMOD input variables are one-dimensional arrays, very few are three-dimensional, and none have more than three dimensions.

In the preceding examples, each added index was associated with a group of resources, from one or more resource categories. A numerical index, such as region, should not be used as an added index, because in this case, there is no straightforward, consistent way to specify weights for aggregation. The preprocessor does allow a special type of index, which we have called a detail index, to be used as an added index for a

variable. This index is associated with a user-supplied list of combat-related names. These names might refer to resources that NAVMOD does not model explicitly, additional important features of combat that can affect appropriate data values for an input, or even code names of different scenarios. Associated with each name is a user-specified relative weight. (For more information on detail indices, see Chapter II, Section B.2.)

c. Additional Comments

The key reason for the entire structure of the aggregator preprocessor is the presence of patterns of regularity and repetition in the elements of the array dimensions of NAVMOD input variables. The different elements of the NAVMOD input array SSRBES, discussed earlier in this section, refer to different types of Blue surface ships. Many NAVMOD inputs have such a dimension on type of Blue surface ship. Similarly, many NAVMOD inputs have a dimension on type of Blue sea-based aircraft, or type of Red surface ship, or some other kind of resource (this can easily be observed in the definitions of NAVMOD input variables given in Appendix F of Reference [2]). For each such dimension, the aggregator preprocessor develops a set or sets of weights and actual resource types (or other combat-related names), which can then be applied to many different variables.

In general, the more disaggregated data have been formed from the NAVMOD data by replacing the NAVMOD notional resources with actual resource types. These data are indeed more disaggregated than the NAVMOD input data, but some manipulation of data from available sources will most likely be needed to obtain data values for the more disaggregated data. The more disaggregated data is still organized by NAVMOD variable name, and many characteristics associated with a NAVMOD variable are also associated with the more disaggregated variable. For example, if a certain NAVMOD vulnerability rate includes a probability of detection by the enemy, then the corresponding more disaggregated data for this variable should also include this probability. If a certain variable represents a sortie rate for aircraft that are already assumed to be available (i.e., the sortie rate is applied to the number of available aircraft), then the more disaggregated data values for this variable should reflect this. The definitions of the NAVMOD variables (in Appendix F of Reference [2]), and the text of References [1] and [2] discuss the particular features incorporated in each NAVMOD variable. Although there are patterns of regularity, no general rule exists concerning which features are included in each variable. The reader

should thus be familiar with the use of a variable in NAVMOD before attempting to obtain more disaggregated data values for the variable.

2. Weights, Resources, and Limit Variables

The preceding section has shown some examples of the type of more disaggregated data that the aggregator preprocessor operates with. The preprocessor computes values for NAVMOD inputs by determining weighted averages of these data. The weights used in the averaging are computed by the preprocessor, based on certain inputs. The structure for computing and managing the weights for aggregation is somewhat complicated, with many different cases and variations; the bulk of Chapter III discusses this structure. The basic idea, however, is to associate each notional type of resource, each type of resource that NAVMOD can model, with a set of actual resource types and relative weights for these actual resource types.[5] For example, a notional type-2 Blue escort ship might be considered, on average, as one-third DDG-993 and two-thirds DDG-51. The same relative weighting would be used for all data to be computed for notional type-2 Blue escort ships.

These relative weights for resources are not directly entered by the user; they are computed based on the values of more disaggregated resource variables. Recall that in the NAVMOD documentation, an informal taxonomy of NAVMOD inputs into effectiveness variables, resource variables, and limit variables was introduced. (See Chapter III and Appendix B of Reference [2].) In the aggregator preprocessor, this taxonomy of variables has been expanded and refined and plays a key role in the structure of the preprocessor's algorithms. Limit variables indicate the number of elements of an array dimension that the NAVMOD code will actually use and typically indicate the number of notional types of a resource to be played, for those resources of which NAVMOD can model multiple types. Resource variables indicate current amounts of the resources NAVMOD plays; initial values are input and the variables are updated as losses and movement occur. A new group of variables, the allocation fractions, affect the relationship between actual resource types and NAVMOD resource types. Data for these variables follow a specific form; they are input to the aggregator preprocessor but not to NAVMOD itself. (Chapter II, Section C.3.c, describes these variables.) The term effectiveness variables is used to indicate the NAVMOD inputs that do not fall into some other group (in a sense, all NAVMOD inputs relate to combat effectiveness). Although the preprocessor computes aggregated values for

---

[5] Throughout this paper, the term "resource" should be regarded as also encompassing ordnance.

each NAVMOD input, different types of variables are treated in different routines of the preprocessor, with different algorithms.

All of the examples of more disaggregated data used in the preceding section were for effectiveness variables. However, the more disaggregated data also include values for resource variables, limit variables, and allocation fraction variables. These variables have special sequences of added indices that are fixed to mesh correctly with the preprocessor algorithms; these indices should be changed by the user only with extreme caution. The particular indices and the form of the more disaggregated data are explained in Chapter II. Also, a few new resource and limit variables have been developed for the aggregator; these are closely related to certain NAVMOD inputs but are not identical to them. Chapter II, Section C.3, discusses these new variables.

The values of the more disaggregated resource variables are used in conjunction with the allocation fractions to compute: 1) aggregated values of the resource variables, suitable for input to NAVMOD; 2) relative weightings for the actual resource types associated with each notional resource type; and 3) certain additional sets of weights, (discussed in Chapter III, Section E). The aggregated values of resource variables are written out on a file in a format readable by NAVMOD. The relative weights for resources are stored in the INGRES database, from which they are later retrieved and used to compute weights for averaging the effectiveness data. After the program is finished running, the relative weights for resources remain stored in the INGRES database, where they can be viewed. The reason that the relative weights for resources are computed, instead of entered directly by the user, is that the more disaggregated resource variables are the source for all three preceding groups of quantities. The algorithms that compute the relative weights for resources from the more disaggregated resource variable values are straightforward but depend on the way the resource is modeled in NAVMOD. (These algorithms are discussed in detail in Chapter III, Section E). The values of the relative weights for resources are controlled by the user but are not directly input by the user. Appropriate choices of values for the more disaggregated resource variables, however, can produce the desired relative weights for resources.

The limit variables require special mention. Chapter II, Section C.3.b, contains a complete list of the limit variables, with definitions, and discusses their use. Frequently, the values of certain limit variables affect which particular NAVMOD array element will be used to hold an aggregated data value; in this sense, the aggregated data values depend on the values of the limit variables. It is therefore required that the values of all limit variables

be decided on prior to running the aggregator preprocessor and not be changed in the NAVMOD data itself (exceptions are discussed in Chapter IV, Section B.2.f). The values of the aggregated resource and ordnance stock variables can be changed in different runs of NAVMOD, with the understanding that each NAVMOD resource type corresponds to a certain average combination of actual resource types.

The preprocessor has two parts. The first processes the resource, ordnance, and limit variables. It computes aggregated values for these variables, and computes and stores in the INGRES database the weights that will be used to aggregate the effectiveness variables. The second part determines (from input) which effectiveness variables are to be aggregated and for each such variable computes the aggregated values. The two parts can be run separately, but if the second part alone is run, the first part must have been run some time previously, so that the weights for averaging have been computed and appropriately stored in the INGRES database.

3. Some Database Structures

To track the more disaggregated data and weights in a consistent manner, the database is organized into several different structures. The following paragraphs summarize how they operate; each structure is explained in detail in the following chapter.

To illustrate how the more disaggregated data and data weights are consistently tracked, the variable SSRBES, (discussed in Section B.1 of this chapter) will be used as an example. SSRBES is a one-dimensional array, where different elements along the dimension correspond to different types of Blue surface ships. In NAVMOD itself, these elements correspond to the notional types of Blue surface ships that NAVMOD can model; in the more disaggregated data, the elements correspond to different actual types of Blue surface ships. Many other variables besides SSRBES have a dimension that encompasses different types of Blue surface ships. The aggregator's structure ensures that whenever a variable has such a dimension, the same list of actual types of Blue ships will occur in the more disaggregated data--the types of actual Blue surface ships considered will depend on only the presence of that certain dimension type, not the variable. Also, the same relative weighting factors for aggregation will be used. For example, if a notional type-2 Blue escort ship is to be considered, on average, as one-third DDG-993 and two-thirds DDG-51 then this weighting will be applied to all data that involve notional type-2 Blue escort ships.

Consistency is ensured by the use of the concepts "index" and "resource category." These are detailed in the following chapter, but, in brief, they operate as follows:

- The more disaggregated data are separated into data for variables, where each variable in the more disaggregated data corresponds to a NAVMOD input variable (with a few exceptions to be discussed).
- Each variable has an associated sequence of dimensions (zero if the variable is a scalar). Some of these dimensions might not be associated with NAVMOD input, but have been added for the more disaggregated variable. Let D denote the total number of dimensions (i.e., the variable is a D-dimensional array).
- The name of (exactly) one index is associated with each dimension.
- Each index is associated with exactly one of the following:
  - -- a resource category or sequence of resource categories (resource-based index; there are several subcases of this);
  - -- the set of integers {1,...,n} for some value of n (numerical indices);
  - -- an ordered list of special resource names or other combat related names (detail index; this special case is discussed in Chapter II, Section B.2--it can be useful but should not be focused on now).
- Each resource category is associated with an ordered set of actual resource types. A sequence of resource categories can be associated with a set of resource types, encompassing (the union of) all of the types in the categories in the sequence.
- Thus, each variable is associated with an ordered family of sets, one set for each index. Each set consists of a list of resource names, or the numbers {1,2,...,n}, for some value of n,[6] or a list of combat-related names. Let the term "component name" denote an actual resource name, a (character-encoded) positive integer, or a combat-related name.
- The cartesian product of the D sets in this family is the set of all D-tuples of component names for that variable. With each D-tuple there is associated a more disaggregated data value, which is set by the user.[7,8]

The preceding discussion has indicated how the structure of the more disaggregated data is linked to the concepts of index and resource category. A similar linkage exists for the weights used for aggregation, as follows.

- A list of actual resource types is associated with each resource category. Sets of weights for these actual resource types are computed and stored in the database. There is one set for each notional resource type. Within each set, there is one weight for each actual resource type on the list, and the weights sum to one.

---

[6] These numbers are encoded into characters; see Chapter III, Section I.1.

[7] See Chapter II, Section E for an exception to this cartesian product structure.

[8] As indicated in Section B.1.b, if a certain D-tuple corresponds to a situation that simply would not occur in reality, inputting a negative data value will cause that D-tuple to not be included in the aggregation process. See Chapter III, Section H.1.

- Associated with each index is a list of component names (which are often actual resource types). Sets of weights for these component names are computed and stored in the database. (Many cases and variations exist, which will be discussed in Chapters II and III.) Of course, for resource-based indices, these weights will be related to the weights for the resource categories. Within each set, there is one weight for each component name on the list, and the weights sum to one.
- As previously indicated, a D-dimensional variable is associated with a family of D sets (lists) of component names, one set for each dimension. For each dimension, the program takes one of the sets of weights that have been computed for the index associated with that dimension.[9] For each D-tuple of component names in the cartesian product, there is then a corresponding product of D weights. This product is used to weight the more disaggregated data value associated with that D-tuple.

Note that the weights are not associated directly with variables. Rather, the weights are associated with indices. Frequently, the same index is used in many variables. The program guarantees that whenever a particular index is used, the appropriate sets of weights for that index are used.

Sometimes NAVMOD models different resources in different ways. For example, NAVMOD models only one notional type of some resources, but an input number of notional types of other resources. Accordingly, the aggregator database distinguishes several different types of resource categories. Similarly, a number of different patterns exist that might be indicated by the elements along an array dimension of a NAVMOD input variable, and thus several different types of indices are distinguished. Chapter II, Sections B.2 and B.3 present a taxonomy of these types. These sections also indicate the information about the resource categories and indices that is stored in the database and the tables where it is stored.

C. USING THE NAVMOD AGGREGATOR PREPROCESSOR COMPUTER PROGRAM

This section explains the mechanics of using the aggregator preprocessor computer program. It is assumed that the program will be run on a VAX/VMS system that has INGRES installed, and that the program user will be registered as an INGRES user. It may also be necessary for the user to be registered as the database administrator of the database accessed by the program (see Chapter IV, Section A). To run the program, the following steps are necessary:

---

[9] The different sets of weights for the indices are used, in turn, for computing the aggregated values of the different array elements of the NAVMOD variable.

- The INGRES database must be set up and filled with the appropriate information.
- The files of more disaggregated data must exist, with the desired data values filled in.
- The source code must be preprocessed, compiled, and linked.
- The appropriate input files must be constructed and associated with the appropriate logical unit numbers.

Each of these items is discussed in the following paragraphs.

1. Preparation of the INGRES Database

The database can be set up using INGRES-supplied procedures (References [6] and [16]). The name of the database should be NAVPRE because this name is referred to in a CONNECT statement at the beginning of the program code. If another name for the database is desired, this statement should be changed.

The database can be delivered with information in it sufficient to run the program. However, the user may wish to alter certain information in the database, most particularly, the actual resource types to be considered and the added indices for variables (see Section B.1.b). To avoid error, the database should be updated according to the procedures discussed in Chapter IV.

The user might need to edit the contents of INGRES table PFNAMEF. This table must have exactly one row in it, and this row must contain the device and directory specification of the files of more disaggregated data (see Section C.2), in the VMS format, with colons and brackets (such as USW:[ESCHWARTZ.PREPROC.AGGR]). All files of more disaggregated data are assumed to reside in this directory. The contents of an INGRES table can be edited with QBF (Reference [12]) or SQL (Reference [5]). If the appropriate device and directory specification are known in advance, the database can be delivered with this specification as the contents of table PFNAMEF.

2. Preparation of Files of more Disaggregated Data

The preprocessor distinguishes several different types of variables, which it operates on in different ways. The more disaggregated data files consist of one file that contains the more disaggregated data values for resource, ordnance, limit, and allocation fraction variables and a set of files that contain the more disaggregated data values for effectiveness variables.

a. Resource, Ordnance, Limit, and Allocation Fraction Data File

The variables in this file will be discussed in detail in Chapter II, Section C.3. An auxiliary program will generate a shell file that contains the correct variables, actual resource types, and other necessary component information, with zeros where the data values should be. The user can fill in the appropriate data values with a text editor. The auxiliary program is discussed in Chapter IV, Section C.6. The file name and extension of the shell file can be renamed, as desired. The file should reside in the device and directory that appears in table PFNAMEF of the INGRES database. (The shell file generated by the auxiliary program will reside in the device and directory specified in table PFNAMEF at the time the auxiliary program is run.)

b. Effectiveness Data Files

There can be many files of more disaggregated effectiveness data. The data for a given effectiveness variable must appear in exactly one file, and the file name and extension of this file should appear in column filname of INGRES table EVARINFO, in the row for that variable. An auxiliary program generates shell files for the effectiveness data, based on a certain categorization of NAVMOD variables. (This program is discussed in Chapter IV, Section C.5.) The user can edit in the appropriate data values with a text editor. The auxiliary program updates the INGRES database (table EVARINFO) to reflect the proper correspondence between the variable names and the data file names. The user can rename the data files (file name and extension) if desired but must take care to then update the INGRES database accordingly. All effectiveness data files should reside in the device and directory that appears in table PFNAMEF of the INGRES database. The shell files generated by the auxiliary program will reside in the device and directory specified in table PFNAMEF at the time the auxiliary program is run.

Some restrictions and suggestions regarding certain more disaggregated effectiveness data values are discussed in Chapter IV.

The amount of disk space occupied by the effectiveness data files depends on the structure of the variables (in particular, on the added indices and actual resource types), but it can be quite large (about 11000 VAX blocks, or 5500K, for one illustrative data set) and the user should make sure that sufficient disk space is available.

3. Preparing the Computer Program

The main source code file for the aggregator preprocessor should be run through the embedded SQL preprocessor, FORTRAN compiler, and linker, as indicated in the chapter on preprocessor operation in the *Embedded SQL FORTRAN Companion Guide* (Reference [10]).[10] Most of the source code of the program is on one large file, but some of the code is on separate files, which are brought into the main source code file by means of INCLUDE statements. Table A-5 of Appendix A indicates the contents of these included files (also see Table A-9). After the necessary input files have been prepared and associated with the correct logical units (as indicated in the next subsection), the program can be run. Since the program may require extensive run time, the user might want to submit it as a batch job.

4. File Structure

This section discusses the input files that are necessary to run the aggregator preprocessor and the output files that it produces.

a. Input Files

The program uses files of more disaggregated data, as mentioned in Section C.2, and two input program control files as follows.

The input options guide file allows the user to choose which part(s) of the aggregator should be run and which variables should be aggregated in the given run of the program. (Section 5 describes how to set up this file.) The file should be associated with logical unit 3 at the command language level. Its contents are read by the main program and/or by Subroutine EFFVAR, into certain program variables.

The information file contains information on the dimensioning and storage of the NAVMOD (aggregated) variables. This file is also used by NAVMOD, and the preprocessor has been designed so that the information file used by NAVMOD can also be used with the preprocessor without modification. The contents of this file are described in Reference [2], Chapter III, Section D.3. The file should be associated at the command

---

[10] INGRES supplies a preprocessor that converts an embedded SQL program into a FORTRAN program (with calls to INGRES routines), which can then be put through the VAX FORTRAN compiler. Except in this subsection, the term "preprocessor" will refer solely to the aggregator preprocessor for NAVMOD.

language level with logical unit 22. (This file, named BLKINP.DAT, is provided with NAVMOD.) Its contents are read by Subroutine GETIVA into several COMMON blocks.

b. Output Files

The program generates a file of informative messages. This file might be especially useful if the program terminates prematurely. The file is associated with logical unit 4; if desired, the user can name it at the command language level or assign it to system output (e.g., the terminal screen).

Currently, the program generates a file that gives the times the preprocessor has taken to perform various operations. This file is associated with logical unit 7, but it might be deleted from future program versions.

The preprocessor has essentially two parts. The first part processes tne resource, ordnance, limit, and allocation fraction variables and computes weights. The second part computes weighted averages of the effectiveness variables.

When the first part of the aggregator is run, the program produces a file of aggregated values for all NAVMOD resource variables, ordnance stock variables, and limit variables. This file is named RESORDLM.AGD; the user might wish to rename it after running the program. The file can be read by NAVMOD.

When the second part of the aggregator is run, the program produces a file of aggregated values for the NAVMOD effectiveness variables indicated in the input options guide file. The output file of aggregated values is named EFFVARS.AGD; the user might wish to rename it after running the program. The file can be read by NAVMOD.

RESORDLM.AGD adjoined with an EFFVARS.AGD file in which every NAVMOD effectiveness variable appears results in a complete set of input data for NAVMOD.

5. Construction of the Input Options Guide File

One of the input files read by the aggregator preprocessor is an input options guide file, containing codes that indicate the specific portions of the aggregator preprocessor the user desires to execute. This section explains how to prepare such a file. The input guide file should be associated with logical unit 3 at the command language level before the

aggregator preprocessor is run. (Note: Only the standard options are discussed here. See Chapter IV, Section A, for some additional options.)

To run only the first part of the aggregator preprocessor, the user must prepare an input file consisting of the following two lines:

0

GCROL000.DAT (or name of other file containing the more disaggregated data for resource, ordnance, limit, and allocation fraction variables).

Several different ways to run the second part of the aggregator preprocessor exist. If the input guide file consists of a single line containing a 1 in the first column, then all effectiveness variables will be aggregated. If the input guide file appears as follows:

2

filename1

filename2

. . .

filename*n*

the effectiveness variables with more disaggregated data in the files filename1 through filename*n* will be aggregated. The "filenames" should contain the file name and extension only; the device and directory are specified in table PFNAMEF of the INGRES database. If the input guide file appears as follows:

3

varname1

varname2

. . .

varname*n*

aggregated values for the variables varname1 through varname*n* will be computed.

To run both the first and the second parts of the aggregator, the input guide file should consist of the following:

0

GCROL000.DAT (or other appropriate file)

(one of the input guide files for the second part, as above)

To run the entire aggregator on all the data, the input guide file should be as follows:

0
GCROL000.DAT (or other appropriate file)
1

Recall that the output files of the aggregator are named RESORDLM.AGD for the resources, ordnance, and limits, and EFFVARS.AGD for the effectiveness variables.

II. STRUCTURES AND CONCEPTS

One aim of this chapter is to explain, in some detail, the structures and concepts that the aggregator is based on. The concepts of resource category, index, and variable are discussed as they apply to the preprocessor. The form of the more disaggregated data for the resource, ordnance, limit, and allocation fraction variables is described.

Another goal of this chapter is to provide the user with sufficient understanding of the INGRES database NAVPRE to be able to change it, if desired. Most of the INGRES tables and their columns are explained (the remainder are discussed in Chapters III and IV). For reference, Appendix B contains a complete listing of the INGRES tables used by the aggregator, with definitions of each column.

The structure of the aggregator database and program evolves from and is consistent with the way NAVMOD models resources and the structure of the NAVMOD input variables. The key fact that underlies and motivates the entire form of the aggregator preprocessor is the presence of patterns of regularity and repetition in the elements of the array dimensions of NAVMOD input variables. For example, many NAVMOD arrays have a dimension on type of Blue surface ship, in which different array elements along this dimension correspond to different notional types of Blue surface ships. The preprocessor should treat this type of dimension in the same manner for all NAVMOD inputs in which it occurs. Ensuring such consistency is the reason for the rather involved series of structures discussed in this chapter. The declaration in the database of many different kinds of resource categories, indices, and variables is necessary to encompass properly the variety of NAVMOD inputs.

A. RESOURCE CATEGORIES

1. Introduction

The resource categories link the actual resource types with the resource types NAVMOD models. Each resource category is associated with one or more NAVMOD variables and also with a set of actual resource types. A resource category has been established for each category of resources that NAVMOD can explicitly model. In this

context, "explicitly" means that: (1) there is at least one NAVMOD input variable or array element that specifies an amount of that resource at the current point in the simulation and (2) the initial value of that variable or array element indicates the initial amount of that resource and the value is updated as changes in the amount of that resource occur. In the aggregator, each resource category is identified by a name, a character string of length 6. Table II-1 lists the 56 resource categories and their names. (Chapter II of Reference [1] and Chapter I of Reference [2] discuss the resources and ordnance types that NAVMOD can model.)

For some resources, NAVMOD models a separate stock of the resource for each region. For example, NAVMOD accounts for separate inventories of Red land-based aircraft on vulnerable and invulnerable airbases. For many types of ordnance, NAVMOD accounts for separate reserve and active stocks. Thus some resource categories have more than one associated NAVMOD variable or array element. Differences in the form of the NAVMOD variables for a resource category affect the form of the more disaggregated data and affect the way the aggregator operates. This topic is discussed in Chapter III, Section E (subroutine RSOWGT.)

Some resources (such as ASW helicopters) are not explicitly modeled by NAVMOD, but modeled implicitly through the use of variables. Resource categories have not been established for these resources, but the aggregator can take into account data for actual types of these resources by means of detail indices (discussed in Section B.2). Note that because of the particular way NAVMOD models Red sea-based aircraft, they are not considered to be a resource category (see the discussion in Reference [1], Chapter III, Section B.2). Air-to-air munitions for Red sea-based aircraft are, however, considered to be a resource category.

Associated with each resource category is the number of types of resource in that category that NAVMOD can model. Often, this number is one, but for some resources, NAVMOD models an input number of types, given by the value of some limit variable.[1,2] A resource category with multiple types has been established, however, only if the NAVMOD code does not distinguish between different notional types of resources within

---

[1] In NAVMOD itself, of course, even if the number of types of a resource is limited, any number of resources of a given type can be modeled.

[2] Most of these limit variables are inputs to NAVMOD; all are discussed in Section C.3.b of this chapter.

Table II-1. RESOURCE CATEGORIES OF THE NAVMOD AGGREGATOR PREPROCESSOR

Name	Code	Description
BAAEW	1	Blue sea-based AEW aircraft
BAASW	1	Blue sea-based ASW aircraft
BAATT	1	Blue sea-based attack aircraft
BAFTR	1	Blue sea-based fighter aircraft
BFBT	1	Blue surface ships—barrier tenders
BFCAR	1	Blue surface ships—carriers
BFESC	2	Blue surface ships—escort ships
BFURG	1	Blue surface ships—URG ships
BLAEW	1	Blue land-based AEW aircraft
BLASW	1	Blue land-based ASW/ASuW aircraft
BLFTR	2	Blue land-based fighter aircraft
BOAAA	1	Blue air-to-air munitions for sea-based aircraft
BOAAF	1	Blue anti-surface munitions for sea-based aircraft
BOAAG	1	Blue air-to-ground munitions for sea-based aircraft
BOAAS	1	Blue ASW munitions for sea-based aircraft
BOBAR	1	Blue munitions for barrier submarines
BOFCM	1	Blue land-attack cruise missiles for surface ships
BOFDA	1	Blue AAD munitions for surface ships
BOFDS	1	Blue SSD munitions for surface ships
BOFFF	1	Blue anti-surface munitions for surface ships
BOFTP	1	Blue ASW munitions for surface ships
BOLAA	1	Blue air-to-air munitions for land-based aircraft
BOLAF	1	Blue anti-surface munitions for land-based aircraft
BOLAS	1	Blue ASW munitions for land-based aircraft
BOSCM	1	Blue land-attack cruise missiles for direct-support submarines
BOSTP	1	Blue ASW/ASuW munitions for direct-support submarines
BSBAR	1	Blue barrier submarines
BSDS	1	Blue direct-support submarines
BUA	1	Blue sonobuoys for sea-based patrol ASW aircraft
BUL	1	Blue sonobuoys for land-based patrol ASW aircraft
BUR	1	Blue sonobuoys for reactive (sea-based) ASW helicopters
RFBT	1	Red surface ships—barrier tenders
RFNRP	2	Red surface ships—non-resupply
RFRSP	1	Red surface ships—resupply ships
RLBMR	2	Red land-based aircraft—bomber
RLFTR	1	Red land-based aircraft—fighter/escort
RLINT	1	Red land-based aircraft—interceptor
RMABD	2	Red land-based SAMs for airbase defense
RMPPD	2	Red land-based SAMs for power-projection defense
ROBAR	1	Red munitions for barrier submarines
ROFAA	1	Red air-to-air munitions for sea-based aircraft
ROFAS	1	Red air-dropped ASW munitions for surface ships
ROFDA	1	Red AAD munitions for surface ships
ROFDL	1	Red long-range SAMs for surface ships
ROFDS	1	Red SSD munitions for surface ships
ROFFL	1	Red long-range ASuW munitions for surface ships
ROFFR	1	Red regular-range ASuW munitions for surface ships
ROFTP	1	Red ASW munitions for surface ships
ROLAA	1	Red air-to-air munitions for land-based aircraft
ROLAF	1	Red anti-surface munitions for land-based aircraft
ROSCM	1	Red attack munitions for missile-firing submarines
ROSTP	1	Red attack munitions for torpedo-firing submarines
RSBAR	1	Red barrier submarines
RSCM	4	Red missile-firing submarines
RSCMA	5	Red missile-firing submarines, aggregated to 1 type
RSTP	1	Red torpedo-firing submarines

that category--only if the NAVMOD code for a notional type-i resource of a given category is of the same form as for a notional type-j resource in that category, with only the value of the array subscript varying (the i-th element of an array being used instead of the j-th element). If the NAVMOD code is different for different elements of an array in at least one place in NAVMOD, then different resource categories have been established corresponding to these different elements. For example, even though inputs such as SSRBES (mentioned in Chapter I, Section B.1), have a dimension that encompasses Blue carriers, escort ships, and URG ships, the NAVMOD code does not treat these three groups of resources identically, and thus different resource categories have been established for them. (The index for the dimension of Blue surface ships is associated with the (ordered) sequence of three resource categories: Blue carriers, Blue escort ships, and Blue URG ships.)

NAVMOD treats Red missile-firing (non-barrier) submarines somewhat differently than other resources. As discussed in Reference [1], Chapter II, Section B.5, NAVMOD models one type of Red missile-firing submarine but with four different subtypes. In some places in the NAVMOD code, all of the subtypes are treated identically; in other places they are treated differently. In the aggregator, a separate algorithm has been developed to handle the case of a resource with multiple subtypes. Two resource categories have been established for Red missile-firing submarines. One treats them as a group; the other treats them as four separate types. (The relevance of this distinction is explained in the discussion of indices.)

Among NAVMOD's resource variables are a number of Red aircraft shelters on vulnerable Red airbases and an amount of runway repair material for vulnerable Red airbases. (For example, see Table B-4 of Reference [2].) NAVMOD models only one type of each of these resources. Currently, resource categories do not exist for these two resources but establishing them would be easy to do with straightforward database changes, as indicated in Chapter IV. (The resources are treated by subroutine RSOWGT, as explained in Chapter III, Section E.) If the user wants to add a dimension on type of shelter or type of runway repair material to some NAVMOD input, the user may wish to make these changes.

Reference [2] (see especially Chapter I, Section B.3), indicates that it is possible to have NAVMOD model the last notional type of Red surface ship as a munitions resupply ship. In the aggregator, it is assumed that this is the case. Separate resource categories

have been set up for Red resupply ships and Red non-resupply surface ships. Certain limit and resource variables used by the aggregator reflect this distinction, which is discussed in Section C.3.b. All aggregated resource and effectiveness data for the last used element along a dimension involving Red surface ships are assumed to apply to resupply ships.[3] If the user does not wish to play resupply ships in NAVMOD, then the NAVMOD inputs can be suitably adjusted after the aggregator is run (see Reference [2]), or the more disaggregated data can be set so that all relevant aggregated data will be zero.

2. Actual Resource Types

Associated with each resource category is a set of actual resource types to be considered as belonging to that category. While the resource categories are (relatively) fixed, based on the kinds of resources NAVMOD can model, the user can specify the actual resource types that are appropriate. The same actual resource type can be specified for two or more resource categories. For example, an F/A18 may be included in both the Blue sea-based attack aircraft category and the Blue sea-based fighter aircraft category. Each actual resource type is identified by a name (alphanumeric string) consisting of up to 10 characters, and an ordering is specified for the set of actual resource types within each category. Table A-8 of Appendix A provides an example of what actual resource types might be included. The information on actual resource types is stored in INGRES table GENTYP0, which has columns for the resource category name, the order number (1,2,3...) of the actual resource type within the resource category, and the name of the actual resource type, as indicated in Section A.3. If the user wishes to change the actual resource types considered, table GENTYP0 can be edited with QBF or SQL (References [12], [6]). Because the resource category names and actual resource type names also appear in several other tables and data files, several auxiliary computer programs have been developed to ensure that when the actual resource types in table GENTYP0 are changed, the other tables and files will be changed appropriately.[4] If GENTYP0 is changed, then these auxiliary programs should be run (see Chapter IV, Section D.2.a).

The number of actual resource types associated with a resource category (the cardinality of the set of actual resource types) is stored in INGRES table GENUSINFO.

---

[3] The last used element is the element corresponding to the value of a certain limit variable; see Section C.3.b of this chapter.

[4] In general, the data in the INGRES data base NAVPRE have not been completely normalized. The aggregator preprocessor and the various auxiliary programs update the database to ensure that data inconsistencies will not occur.

3. Relevant INGRES Tables

The INGRES tables GENUSINFO, GENTYP0, and RESGENUS contain basic information about the resource categories, most of which is set (by the user and by certain auxiliary programs) before the aggregator is run.[5] Table GENUSINFO (mnemonic for genus information) contains the information indicated in Table II-2. Regarding the code numbers for resource categories (column genuscode), NAVMOD does not contain any code-3 resource categories, but the aggregator methodology can process them. The only code-4/code 5 resource category is Red missile-firing submarines in regions (see Reference [1], Chapter II, Section B.5).

Table GENTYP0 (mnemonic for genus types initial) contains the actual weapon types associated with each resource category. The columns of table GENTYP0 and their meanings are shown in Table II-3; note that the actual weapon types in each category appear in an ordered sequence. Table A-8 of Appendix A contains lists of actual resource types that might be appropriate for these resource categories; see also Chapter IV, Section B.1. Table RESGENUS (mnemonic for resources and genuses) gives the names of the NAVMOD resource or ordnance variables that are associated with the resource categories and certain additional information, as shown in Table II-4. Table RESGENUS is accessed by Subroutine RSOWGT (described in Chapter III, Section E). The information in table RESGENUS evolves directly from the NAVMOD structure itself; the table needs no change unless NAVMOD changes and would have to be changed if NAVMOD's treatment of resources changed.

---

[5] The names for the database tables and columns frequently contain the character string "genus"; this string is intended to refer to resource category and should be thought of as such. (One could consider the individual actual resource types to be "species" within the resource category "genus.")

Table II-2. COLUMNS OF INGRES TABLE GENUSINFO AND THEIR MEANINGS

Column Name	Type and Length	Meaning
genusnam	character 6	code name of resource category
genuscode	integer 1	code number of resource category, as follows: 1--NAVMOD simulates one notional type of this resource 2--number of types of this resource that NAVMOD simulates is given by a limit variable 3--NAVMOD simulates fixed number of types of this resource 4--NAVMOD simulates fixed number of subtypes of this resource; each subtype can have differences in its modeling 5--Resource category corresponds to an aggregation of subtypes in some code-4 category
nrltyp	integer 2	Number of actual weapon types in this resource category (see description of INGRES table GENTYP0)
nntltyp	integer 2	Number of notional types of this resource that NAVMOD simulates (if this number is given by the value of a limit variable (genuscode=2 case), then the aggregator fills in this column with the value input by the user for that limit variable)
ldnam	character 6	Name of limit variable (for code-2 resource categories) or name of index, if any, (for code-3 or code-4 resource categories)
side	character 4	Blue or Red
genusmemo	character 47	brief description of this category

Table II-3. COLUMNS OF INGRES TABLE GENTYPO AND THEIR MEANINGS

Column Name	Type and Length	Meaning
genusnam	character 6	code name of resource category
ordno	integer 2	order number of actual weapon type within this resource category
rltyp	character 10	10-character code name of actual weapon type

Table II-4. COLUMNS OF INGRES TABLE RESGENUS AND THEIR MEANINGS

Column Name	Type and Length	Meaning
imeth	integer 1	code number of aggregation method for resource category
resnam	character 6	name of main NAVMOD resource variable (for which a notional value is to be computed)
genusnam	character 6	code name of resource category
adlvar	character 6	name of additional NAVMOD variable that may be relevant--the specific use of such variable depends on the aggregation method
imuni	integer 1	further code number for aggregation method

Table GENUSTYWGT (mnemonic for genus types and weights) contains, for each resource category, relative weights for the actual weapon types that correspond to each notional type that NAVMOD will simulate. Table II-5 shows the columns of table GENUSTYWGT and their meanings. The information in the first four columns of table GENUSTYWGT, on actual and notional resource types, is in the table before the aggregator is run.[6] The weights are computed by the aggregator and stored in columns rawwgt and rltypwgt.

B. INDICES

1. Introduction

In this documentation, the term "index" is used to mean a name for a dimension of an array variable. This name is used as an identifier for a number of different quantities, lists, and other entities that relate to that dimension, as it is used in NAVMOD and/or the more disaggregated data. The indices provide the primary link between the input data for NAVMOD and the more disaggregated data.

The use of indices in the aggregator arose out of the treatment of symbolic constants in NAVMOD. (To review the discussion of NAVMOD symbolic constants, see Reference [2], especially Chapter III, Section D.4, and Appendix C.) In NAVMOD, the symbolic constants that have been defined are all used as, in effect, dimension names. Their values indicate the sizes of the dimensions of NAVMOD arrays. These values have been calculated in a consistent manner, and the symbolic constants have been used as dimension declarators in a consistent manner, in accordance with the definitions of the inputs. For example, consider the NAVMOD input SSRBES, mentioned in Chapter I. In NAVMOD, the different elements of this array indicate effectiveness values for different notional types of Blue surface ships. In the NAVMOD code, this variable is declared as SSRBES(MZKBSS). The entity MZKBSS is declared as a symbolic constant, and its value indicates an upper limit on the number of notional types of Blue surface ships to be

[6] If the user wants to change the actual resource types to be considered, the new types should be entered into table GENTYP0. An auxiliary program has been developed that generates appropriate new information for table GENUSTYWGT from the information in GENTYP0. This program, named GENGENUST, is described in Chapter IV.

Table II-5. COLUMNS OF INGRES TABLE GENUSTYWGT AND THEIR MEANINGS

Column Name	Type and Length	Meaning
genusnam	character 6	code name of resource category
ntlindval	integer 2	notional resource type played (ranges from 1 to the number of notional types of this resource that NAVMOD simulates (see INGRES table GENUSINFO)) A value of zero indicates a total over all notional resource types.
ordno	integer 2	order number of actual weapon type (see INGRES table GENTYP0)
rltyp	character 10	code name of actual weapon type (see INGRES table GENTYP0)
rawwgt	float 4	raw weight for indicated actual weapon type that is considered to be notional type ntlindval (computed by program based on sum of appropriate resource variables)
rltypwgt	float 4	normalized weight for indicated actual weapon type that is considered to be notional type ntlindval (computed by program from column rawwgt)

simulated.[7] This symbolic constant is used not only in SSRBES, but also in the declarations of all NAVMOD variables that involve Blue surface ships. The symbol MZKBSS is used as a symbolic constant and as the name for a dimension on type of Blue surface ship.

The form of the more disaggregated data is much the same as that of the NAVMOD inputs. The more disaggregated data are organized into variables that correspond to the NAVMOD inputs; each such variable can be regarded as an array (scalars being considered as zero-dimensional arrays). Each dimension of this array has an associated dimension name; in the aggregator preprocessor, these dimension names are not used as symbolic constants, hence the term "index." (The terms "index" and "index name" will be used synonymously.) If the associated NAVMOD input is d-dimensional, the dimensions of the more disaggregated variable encompass the dimensions of the NAVMOD input, and the

[7] This value was computed by the procedure discussed in Appendix C of Reference [2]. The auxiliary file DCOMMN.DAT referred to in the appendix contains the names and dimension declarations of all the NAVMOD inputs.

symbolic constant names used in the declaration of the NAVMOD variable are the first d indices of the more disaggregated variable.[8] The symbolic constant names of NAVMOD are thus also used as index names in the preprocessor. As indicated in Chapter I, Section B.1.b, the user can let the more disaggregated variable have more dimensions than the associated NAVMOD variable. Each such added dimension must be given an index name, and these names must be set up in the database.[9] For a given variable, the terms "NAVMOD index" and "notional index" denote those indices associated with dimensions of the variable as it is used in NAVMOD; "added index" or "non-notional index" denotes an index associated with an added dimension.

The motivation for the use of indices is that the same (variation of the basic) aggregation algorithm and the same sets of weights are used for a given index, for each variable in which that index appears.[10] This ensures consistency and relative efficiency in the aggregation process.

The remainder of this section discusses the taxonomy of indices, the information associated with indices, and the major INGRES tables that concern indices. Ultimately, indices are associated with lists of component names and associated weights, which are used to average the appropriate more disaggregated data values. Many different cases and conditions exist, however, and this chapter and Chapter III explain these associations.

2. Taxonomy of Indices

The aggregator preprocessor categorizes the indices by index code numbers.[11] The index code numbers, with brief descriptions, are shown in Table II-6. The particular numbers used for index codes have been essentially arbitrarily assigned, but the aggregator computer code does refer to these particular numbers. Different types of indices are, in general, processed with different variations of the aggregation algorithm. This

[8] Most dimensions in the NAVMOD variables are declared by symbolic constants, but a few are declared by numerical values. Since the aggregator requires every dimension to have a name, some index names have been invented for these dimensions, as indicated in Section B.2 of this chapter.

[9] As indicated in paragraphs that follow, a number of added indices have already been developed, and the database can be delivered with these added indices properly registered in it.

[10] An exception to this rule is as follows. Some index names can be used as either notional indices or added indices. Different aggregation algorithms are used for these two cases. This is discussed further in Chapter III.

[11] The code number of an index is stored in column indxcode of INGRES table INDXBINFO, in the row for that index.

Table II-6. BRIEF DESCRIPTIONS OF INDEX CODES

Code Number	Description
2	Single resource category with number of notional types indicated by associated limit variable (resource category name appears in INGRES table INDXBINFO)[a]
3	Single resource category with fixed number of notional types (resource category name appears in INGRES table INDXBINFO)[a]
4	Sequence of resource categories, with fixed total number of notional types (the names and order of these resource categories appear in INGRES table INDXGENUS)
6	Sequence of resource categories; number of notional types determined as sum of numbers of notional types of the individual categories in the sequence (the names and order of these resource categories appear in INGRES table INDXGENUS)
10	Adaptive index encompassing two or more resource categories; particular category used depends on value of some other base index (appropriate information is obtained from INGRES tables SPLITINDG and SPLITINDX)
12	Simple numerical index with associated limit variable that indicates the number of elements to be used
14	Simple numerical index with fixed number of elements
16	Compound numerical index--sequence of simple numerical indices (the indices in the sequence appear in INGRES table CMPLIM)
20	Single resource category, when used as non-notional index (resource category name appears in INGRES table INDXBINFO)[a]
30	Detail index--resource types or descriptions correspond to level more detailed than NAVMOD resources; user must supply relative weights (information stored in INGRES table INDXDTYP)
40	Indices used only as second indices of the allocation fraction variables--used in auxiliary programs only

[a]Even when an index corresponds to a single resource category, the index is not considered the same entity as the resource category, and the name of the index is not the same as the name of the resource category (although frequently the two names are similar).

section describes the major distinctions between different types of indices. The details of the aggregation algorithms are discussed in Chapter III, Section H. The taxonomy of indices arises from the following:

- The different ways array inputs are dimensioned in NAVMOD and the various meanings of elements along an array dimension in the NAVMOD inputs;
- The structure of added indices for the resource and ordnance stock variables; and
- Options for added indices that might be useful.

The aim of the aggregator preprocessor is to construct, from the more disaggregated data, data suitable for input to NAVMOD. Thus every type of dimensioning in a NAVMOD input must be treatable by the aggregator. Accordingly, a different type of aggregator index exists for every type of NAVMOD dimension. These types are identified with codes 2, 3, 4, 6, 12, 14, and 16.

Recall from Chapter I that the resource and ordnance stock variables have been given specific added indices that are explicitly referred to in the aggregation algorithms and should not be changed by the user. These indices have been set up in the database and have been given (arbitrarily) the code number 20. Each such index is associated with a single resource category, and each resource category has an associated code-20 index.

For effectiveness variables, the use of added indices can possibly allow the more disaggregated data used by the aggregator to be closer to data from available sources. Several different types of added indices for effectiveness variables are treatable by the aggregator.[12] That is, several different aggregation algorithms have been developed, keyed to the use of different types of added indices. The user can invent added indices as long as they fall into one of these types.[13] Several examples of possible added indices have been put in the database. Of course, the user need not specify any added indices for an effectiveness variable. Developing algorithms to treat additional types of added indices may be possible.

One major distinction is among resource-based indices, numerical indices, and detail indices. Each index has an associated list of component names (which are used to

[12]The details of the aggregation algorithms are explained in Chapter III, Section H.

[13]Those invented indices, of course, must be registered in the database. A row of INGRES table INDXBINFO must be set up for each index, and other tables must be adjusted as appropriate. Details are presented in this section and in Section B.3.

identify the more disaggregated data values). For resource-based indices, the component names are actual resource types, spanning a sequence of one or more resource categories. For numerical indices, they are numbers (encoded into characters; see Chapter III, Section I.1). For detail indices (index code 30) the list of component names is supplied by the user and is stored in the database.

As is evident from the definitions of the NAVMOD inputs, many of the NAVMOD dimensions indicate type of resource. Accordingly, the symbolic constants used as dimension declarators for these dimensions have been set up in the database as resource-based indices. Sometimes, the resources specified are within a single resource category; more frequently, they span two or more resource categories (compound resource-based indices). Indices with codes 2, 3, 4, 6, and 20 are resource-based indices; Table II-6 indicates the distinctions among these codes. (The associated resource category or categories are listed in the database, as explained in Section B.3.)

Resource-based indices (including code-20 indices) can be specified as added indices for effectiveness variables. The aggregation algorithm used when a given index is an added index differs from the algorithm used when that index is a NAVMOD index.

The resource-based indices used as NAVMOD indices encompass many commonly used sequences of resource categories. However, an index can be invented for any sequence of resource categories the user desires, as long as it is declared properly in the database. For example, some NAVMOD effectiveness variables apply to combat against Red submarines by Blue ASW aircraft, but the NAVMOD variable does not allow separate values for land-based and sea-based ASW aircraft. If this distinction were important, the user could invent a code-6 index encompassing these two resource categories and use it as an added index for those variables. (The index name would have to be declared in INGRES table INDXBINFO and the resource categories in table INDXGENUS.) The more disaggregated data would then contain separate data values for land-based and sea-based ASW aircraft; the aggregator would compute an average value to be entered as input to NAVMOD. (An alternative action would be to reprogram NAVMOD.)

Indices with codes 12, 14, and 16 are numerical indices. For effectiveness variables, no attempt is made to average data corresponding to different values of numerical

indices.[14] Numerical indices should not be specified as added indices, as this would then necessitate such averaging.

As indicated earlier, most dimensions in the NAVMOD variables are declared by symbolic constants, but a few are declared by numerical values. Since the aggregator requires every dimension to have a name, some index names have been invented for these dimensions. These index names are shown in Table II-7; all of them have index code 14.

As an option, the aggregator preprocessor allows detail indices, identified by index code 30. The user can invent detail indices as desired, and can use them as added indices for whatever variables are deemed appropriate. For each detail index, the user must specify a series of combat-related names and associated relative weights. This information is stored in INGRES table INDXDTYP (explained in Section B.3.b, below).[15] The combat-related names can be specified by the user (with the current formatting of the database, each name must be 10 characters or less). Detail indices are used to allow the preprocessor to recognize combat detail at finer levels than NAVMOD models. For example, NAVMOD models Red sea-based fighter aircraft and Blue and Red ASW helicopters only implicitly, by means of certain effectiveness variables. The user can invent detail indices on actual types of such resources and use these as added indices for those variables. The more disaggregated data would then contain appropriate effectiveness data values for each different actual type of resource; these would be averaged in accordance with the user-supplied weights. These weights could be changed (via the INGRES QBF or SQL utilities) for a different run of the aggregator.

Detail indices might also be used for combat-related factors not directly related to resource type (such as type of tactics or posture) or even as flags for different scenarios (change the relative weights to 1 for the desired scenario, 0 for all others). For example, variables that model sonar effectiveness might depend on water condition and therefore might differ in various parts of the world. A detail index for ocean could be set up and used as an added index for these variables. The combat-related names specified in table INDXDTYP would be the names of the various oceans under consideration. The more disaggregated data, kept on file, could then accept different values corresponding to

[14]The computation of some weights for aggregation involves adding resources over different regions, et al. See Chapter III, Section E.

[15]The index name must also be entered in INGRES table INDXBINFO, as indicated in Section B.3.a.

Table II-7. NUMERICAL INDICES THAT CORRESPOND TO NUMERICAL VALUES IN THE NAVMOD INPUT DIMENSIONING

Index Name	Numerical Value[a]	Comments
MY2	2	Used in several different variables
MY3	3	Used in several different variables
MY5	5	Used in variables RARBAB and RARCAB
MY6	6	Used in variables CVREPP and URGEP
MY10	10	Specified as index for code–25 variable OVRRG; not actually accessed
MY11	11	Specified as index for code–25 variable OVBTF; not actually accessed.
MYCTFI	2	Index in several variables used in Subroutine CTFMOD. Corresponds to I in definitions of DIT, D2T, DLIMR, et al.
MYCTFM	2	Index in several variables used in Subroutine CTFMOD. Blue defense against: 1--Red aircraft; 2--Red missiles after launch
MYCV	4	Third index of variable D2T; value can be reset by procedure of Appendix C of Reference [2].
MYIAB	2	1--vulnerable Red airbases; 2--invulnerable
MYMBLA	2	Blue land-based aircraft on: 1--ASuW attack; 2--escort for such attack (variables EAAKDA, EAFSRQ, EAVSRQ)
MYMBSA	2	Blue sea-based aircraft on: 1--ASuW attack; 2--escort for such attack (used in several variables used by Subroutine SHPSHP)
MYMM	2	1--minimum; 2--maximum
MYMRLA	2	Used as first index of variable PKAT1. 1--Red bombers; 2--Red escort aircraft
MZKBA1	3	Used as index for variable PIWACM. 1--Blue attack aircraft; 2--Blue fighter aircraft; 3--Blue cruise missiles

[a]This value appears in INGRES table INDXBINFO, in columns mxvalntl, ivalntl, and nsteps, in the row for the index in question.

different oceans. Just before the aggregator was run, the relative weights in table INDXDTYP would be edited in accordance with the desired scenario. Additional information on detail indices appears in Chapter III, Section B.

The aggregator also allows adaptive indices, identified by index code 10. The user can invent such indices as desired and use them as added indices for effectiveness variables. Some restrictions apply to the use of adaptive indices, and they are best understood after a discussion of the regular aggregation algorithm. (These restrictions are explained in (Chapter II) Section E, and in Chapter III, Section H.3.b.) It is emphasized that the use of adaptive indices is an option. They can result in some time savings in certain cases, but equivalent aggregation operations can be performed without them.

The code-40 indices have been explicitly set up as part of the resource and weight computation process and should not be changed by the user; they are explained in Section C.3.c.

3. Information About Indices--Relevant INGRES Tables

The INGRES database holds a number of pieces of information concerning indices. Much of this information is preset before the aggregator preprocessor is run.[16]

INGRES table INDXBINFO (mnemonic for indices--basic information) contains certain information about the indices. Each index has exactly one row of table INDXBINFO devoted to it. Other INGRES tables contain information relevant to certain types of indices. This section explains most of these tables; additional tables relating to indices are discussed elsewhere in the text. Appendix B contains a complete list of INGRES tables used by the aggregator preprocessor, with explanations of their columns.

a. INGRES Table INDXBINFO

The columns of INGRES table INDXBINFO and their meanings are shown in Table II-8. The value in column ivalrl (mnemonic for real value) gives the total number of actual weapon types associated with this index. For resource-based indices (codes 2, 3, 4,

[16]The database can be delivered ready to run the aggregator subject to certain assumptions (on specific added indices and actual resource types). If it is desired to change these assumptions, the database and/or the more disaggregated data files must be updated in an appropriate manner. There are auxiliary programs, described in Chapter IV, that aid such updating.

Table II-8. COLUMNS OF INGRES TABLE INDXBINFO AND THEIR MEANINGS

Column Name	Type and Length	Meaning
indxnam	character 6	name of index
indxcode	integer 1	code number of index (see Table II-6)
ivalrl	integer 2	number of actual weapon types associated with this index
mxvalntl	integer 2	upper limit on the numerical value associated with this index in the NAVMOD program itself (value of associated symbolic constant; maximum number of notional types of resources associated with this index)
ivalntl	integer 2	actual numerical value associated with this index in the NAVMOD program itself (value of associated limit variable; number of notional types of resources associated with this index)
nsteps	integer 1	number of distinct resource categories or "component" indices associated with this index
genus	character 6	code name of associated resource category (if there is exactly one such)
indxrnam	character 6	name of "related" index (used if indxcode=40)

6, and 20), this value has been computed from the information in table GENTYP0 (see Section A.3) by the auxiliary programs GENGENUST and GENIXRTYP, before the aggregator itself is run. If the information in table GENTYP0 changes, column ivalrl must be updated appropriately by the auxiliary programs or by hand.

For code-30 detail indices, the value in column ivalrl is interpreted as the number of different combat-related names associated with the index, and is computed by the aggregator in Subroutines STINDD (see Chapter III, Section B), from information in INGRES table INDXDTYP. Table II-9 shows the structure of table INDXDTYP, which is applicable only to code-30 indices. The user must preset the information in columns indxnam, ordno (order number), rltyp (real (actual) type), and rawwgt (raw weight);

Table II-9. COLUMNS OF INGRES TABLE INDXDTYP AND THEIR MEANINGS

Column Name	Type and Length	Meaning
indxnam	character 6	name of index
ordno	integer 2	order number of actual weapon type
rltyp	character 10	10-character code name of actual weapon type
rawwgt	float 4	relative weight for this actual weapon type (must be supplied by user)
rltypwgt	float 4	normalized weight for this actual weapon type (computed by program from preceding column)

Subroutine STINDD computes the values (normalized weights) in column rltypwgt. Column ivalrl of table INDXBINFO is not used for numerical, code-10, or code-40 indices.

NAVMOD frequently distinguishes the value of the symbolic constant used as a dimension declarator from the value of an associated limit variable that indicates the number of elements along the array that will actually be used by the computer program (see Reference [2], Chapter III, Section B). For example, the symbolic constant MXLOC, the maximum number of regions that can be played (without recompiling NAVMOD) is a different entity from the (input) limit variable NLOC, the number of regions to be played, and the value of MXLOC can differ from the value of NLOC. This distinction has been maintained in the aggregator preprocessor--both the NAVMOD symbolic constant value and the limit variable value (if any) are considered to be quantities associated with the index. These quantities are stored in columns mxvalntl (maximum notional value) and ivalntl (notional value) of table INDXBINFO. These quantities do not have meaning for indices that are not used in NAVMOD itself, in particular, those indices with codes 10, 20, 30, and 40, and columns mxvalntl and ivalntl are not used for such indices. For indices used in NAVMOD

- The value in column mxvalntl is preset to the value of the corresponding NAVMOD symbolic constant (the auxiliary program GENMXVAL aids in this);

- For indices that do not make use of a limit variable (index codes 3, 4, 14) the value of column ivalntl is preset to the value of column mxvalntl;
- For indices that make use of a limit variable (index codes 2, 6, 12, and 16) the value of column ivalntl is computed by Subroutines LMVSET and LMVCMP of the aggregator preprocessor, based on certain inputs, as discussed in Chapter III.

For resource-based indices, column nsteps (number of steps) gives the number of resource categories associated with the index. The value in column nsteps must be preset for each index when that index is entered in table INDXBINFO (and has been preset correctly for all indices currently there). For indices with codes 2, 3, and 20 (single resource category indices) the value in column nsteps should equal 1, and the name of the corresponding resource category is shown in column genus. These index codes are the only ones for which column genus is applicable. For compound resource-based indices (codes 4 and 6), column nsteps gives the number of resource categories associated with the index. The particular sequence of resource categories is listed in INGRES table INDXGENUS (mnemonic for indices and genuses), as shown in Table II-10. INGRES table INDXGENUS is applicable to code-4 and code-6 indices only; the information in it must be (has been) preset appropriately.

For numerical indices (codes 12, 14, and 16) the value of column nsteps is set equal to the value of column ivalntl. For code-14 indices, this is preset; for code-12 and code-16 indices, database updates are performed by Subroutine LMVSET and LMVCMP, respectively.

Column indxrnam (related index name) is applicable to code-40 indices only, which are not used by the aggregator preprocessor itself, but are used by the auxiliary program GENROLDAT, as discussed in Chapter IV. For these indices, the information in column indxrnam has been preset.

b. Other INGRES Tables That Concern Indices

Two additional INGRES tables, SPLITINDG and SPLITINDX, apply to code-10 indices, and are discussed in Section E. Table INDXMSG is discussed in Section B.3.c.

The final main table of interest here is INDXRTYP (mnemonic for indices and real (actual) types); the structure of this table is shown in Table II-11. INDXRTYP contains the names of the actual weapon types and weights associated with resource-based indices that have more than one associated resource category (i.e., code-4 and code-6 indices).

Table II-10. COLUMNS OF INGRES TABLE INDXGENUS AND THEIR MEANINGS

Column Name	Type and Length	Meaning
indxnam	character 6	name of index
istep	integer 1	step number (1,2,...)
stepgenus	character 6	code name of resource category associated with step istep of index indxnam
steplvval	integer 2	number of notional types associated with the above resource category (see table GENUSINFO)

Table II-11. COLUMNS OF INGRES TABLE INDXRTYP AND THEIR MEANINGS

Column Name	Type and Length	Meaning
indxnam	character 6	name of index
ordno	integer 2	order number of actual weapon type
rltyp	character 10	10-character code name of actual weapon type
rawwgt	float 4	relative weight for this actual weapon type (computed by program)
rltypwgt	float 4	normalized weight for this actual weapon type (computed by program)

For a given such index, the set of weapon types is the union of the sets of weapon types associated with each resource category associated with this index (see table INDXGENUS). This union has been computed by the auxiliary program GENIXRTYP from the information in INGRES tables INDXGENUS and GENTYP0. The information in columns indxnam, ordno, and rltyp has been pre-computed; the aggregator computes appropriate values for columns rawwgt and rltypwgt in Subroutine INDWGT, as explained in Chapter III, Section F.

Some numerical NAVMOD dimensions (identified in the aggregator by code 16) can be regarded as having a compound structure in that the total number of elements used is given by some sum of limit variable values and/or symbolic constants associated with other indices. INGRES table CMPLIM (mnemonic for computed limits) serves the same purpose for these compound numerical indices as table INDXGENUS serves for

Table II-12. COLUMNS OF INGRES TABLE CMPLIM AND THEIR MEANINGS

Column Name	Type and Length	Meaning
indxnam	character 6	name of index
istep	integer 1	step number (1,2,3,...)
stepindx	character 6	name of index corresponding to this step
stepixval	integer 2	maximum value (dimension limit) for index stepindx
steplvam	character 6	limit variable (if any) corresponding to the index for this step; if no such limit variable, stepindx itself is used
steplvval	integer 2	value of the limit variable in preceding column, or stepixval if no such limit variable

compound resource-based indices. The structure of table CMPLIM is shown in Table II-12. The information must be (has been) preset for all code-16 indices, except for those entries in column steplvval whose corresponding entry in column steplvnam is a limit variable name. This value is filled in by Subroutine LMVSET. (For more information on table CMPLIM, see the description of Subroutine LMVCMP in Chapter III, Section D.)

4. Component Names

A list (ordered set) of component names is associated with every index. These component names, each of which is an alphanumeric string of (up to) 10 characters, help characterize the more disaggregated data, are used by the auxiliary programs that generate the shell files for the more disaggregated data, and play a pivotal role in matching the weights for aggregation with the appropriate more disaggregated data values. (This connection is described in Chapter III.) This section indicates the nature of the component names and the INGRES tables in which they are stored. The specifics differ among the various types of indices.

For code-10 adaptive indices, the structure of the component names is a somewhat complicated; the specifics are discussed in Section E. For code-30 detail indices, the component names are simply the combat-related names specified by the user in INGRES table INDXDTYP. The number of such names associated with a code-30 index is

computed by Subroutine STINDD and stored in column ivalrl of INGRES table INDXBINFO (in the row for that index).

Each code-20 index is associated with exactly one resource category (which is indicated in column genus of table INDXBINFO, in the row for the particular index under consideration). The component names are simply the actual resource types associated with this resource category; these appear, in ordered sequence, in table GENTYP0. The number of actual resource types is stored in column ivalrl of INGRES table INDXBINFO (in the row for that index).

Indices with codes 12, 14, and 16 are numerical indices, and their associated component names are, in essence, the integers from 1 through an appropriate upper limit. For consistency, the component names are these integers encoded into character-length-10 variables. The encoding is performed by Subroutine CHRINI, which is discussed in Chapter III, Section I.1. The appropriate upper limit is the number of array elements along a dimension specified by that index that it is desired to consider. In the aggregator itself, this upper limit is obtained from column ivalntl of table INDXBINFO. For code-14 indices, this is fixed and preset, for code-12 indices this limit is given by the user-supplied value of some limit variable, and for code-16 indices, it is some specified sum of user-supplied values and fixed values. In the auxiliary programs GENEFFDAT and GENROLDAT, the upper limit is obtained from column mxvalntl of table INDXBINFO, as limit variable values are not known when these programs are run.

The remaining types of indices, with code numbers 2, 3, 4, or 6, are resource-based--associated with resource categories. A distinction is made between single and compound resource-based indices, according to whether the number of associated resource categories is 1 or greater than 1. (This number is found in column nsteps of table INDXBINFO.) For single resource-based indices, the component names are simply the actual resource types associated with the resource category associated with the index (just as in the case of code-20 indices). These actual resource types appear, in ordered sequence, in table GENTYP0.

For compound resource-based indices, the set of component names is the union set of all the actual resource types in the different resource categories associated with the index. This set appears in table INDXRTYP. Note that the union set is used by the auxiliary programs and by the aggregator when compound resource-based indices are used as added indices. When compound resource-based indices appear in the NAVMOD indexing of a

variable, the union set is not used as such; a somewhat different procedure is used to determine the appropriate component names. (This procedure is discussed in Chapter III, Section H.)

For both single and compound resource-based indices, the number of actual resource types (in the single associated resource category or in the union set) is stored in column ivalrl of INGRES table INDXBINFO (in the row for the particular index under consideration).

In various routines of the aggregator, the computer code needs to know the component names associated with a given index. The information in table INDXBINFO for that index lets the code know which treatment is to be used. If the index is non-numerical, the code retrieves the component names from the appropriate INGRES table. If the index is numerical, the code retrieves the appropriate upper limit from column ivalntl of table INDXBINFO (in the row for the index under consideration) and generates the component names as the character-encoded versions of the integers from 1 through this upper limit.

5. Listing of Indices

Table II-13 presents a complete list of indices currently set up in the aggregator database. This list encompasses all index names used for NAVMOD dimensions, one code-20 index for each resource category, the necessary code-40 indices, and some illustrative code-6, code-10, and code-30 indices that could possibly be used as added indices for certain variables. Currently, index names that are used as NAVMOD dimensions start with the letter M, and index names that would be used only as added indices begin with the letter J, but the aggregator code does not make use of this fact; any character string of length 6 or less can be used as an index name.

The descriptions in Table II-13 are stored in the INGRES table INDXMSG. This INGRES table is not used by the aggregator itself, but has been established to have descriptions of indices on-line and to facilitate the development of reports. The structure of INGRES tables INDXMSG is shown in Table II-14.

Table II-13. ALPHABETICAL LISTING OF AGGREGATOR INDICES

Index Name	Code Number	Description
JABCML	30	Blue method of launching land-attack cruise missiles
JABDSM	30	Blue submarine screen protocol
JABHEL	30	Blue sea-based ASW helicopter type
JARHEL	30	Red sea-based ASW helicopter type
JARSBA	30	Red sea-based (fighter) aircraft
JBAAEW	20	Blue sea-based AEW aircraft
JBAASW	20	Blue sea-based ASW aircraft
JBAATT	20	Blue sea-based attack aircraft
JBAFTR	20	Blue sea-based fighter aircraft
JBFBT	20	Blue surface ships—barrier tenders
JBFCAR	20	Blue surface ships—carriers
JBFESC	20	Blue surface ships—escort ships
JBFURG	20	Blue surface ships—URG ships
JBLAEW	20	Blue land-based AEW aircraft
JBLASW	20	Blue land-based ASW/ASuW aircraft
JBLFTR	20	Blue land-based fighter aircraft
JBOAAA	20	Blue air-to-air munitions for sea-based aircraft
JBOAAF	20	Blue anti-surface munitions for sea-based aircraft
JBOAAG	20	Blue air-to-ground munitions for sea-based aircraft
JBOAAS	20	Blue ASW munitions for sea-based aircraft
JBOBAR	20	Blue munitions for barrier submarines
JBOFCM	20	Blue land-attack cruise missiles for surface ships
JBOFDA	20	Blue AAD munitions for surface ships
JBOFDS	20	Blue SSD munitions for surface ships
JBOFFF	20	Blue anti-surface munitions for surface ships
JBOFTP	20	Blue ASW munitions for surface ships
JBOLAA	20	Blue air-to-air munitions for land-based aircraft
JBOLAF	20	Blue anti-surface munitions for land-based aircraft
JBOLAS	20	Blue ASW munitions for land-based aircraft
JBOSCM	20	Blue land-attack cruise missiles for dir.-supt. submarines
JBOSTP	20	Blue ASW/ASuW munitions for direct-support submarines
JBSBAR	20	Blue barrier submarines
JBSDS	20	Blue direct-support submarines
JBUA	20	Blue sonobuoys for sea-based patrol ASW aircraft
JBUL	20	Blue sonobuoys for land-based patrol ASW aircraft
JBUR	20	Blue sonobuoys for reactive (sea-based) ASW helicopters
JCCAEW	6	Blue AEW aircraft—land-based plus sea-based
JCCASW	6	Blue (fixed-wing) ASW aircraft—land-based plus sea-based
JDOFST	10	Blue torpedo—surface- or sub-launched. Keyed to MZKBS.
JRFBT	20	Red surface ships—barrier tenders
JRFNRP	20	Red surface ships—non-resupply
JRFRSP	20	Red surface ships—resupply ships
JRLBMR	20	Red land-based aircraft—bomber
JRL[illegible]TR	20	Red land-based aircraft—fighter/escort
JRLINT	20	Red land-based aircraft—interceptor
JRMABD	20	Red land-based SAMs for airbase defense
JRMPPD	20	Red land-based SAMs for power-projection defense
JROBAR	20	Red munitions for barrier submarines
JROFAA	20	Red air-to-air munitions for sea-based aircraft
JROFAS	20	Red air-dropped ASW munitions for surface ships
JROFDA	20	Red AAD munitions for surface ships
JROFDL	20	Red long-range SAMs for surface ships
JROFDS	20	Red SSD munitions for surface ships
JROFFL	20	Red long-range ASuW munitions for surface ships
JROFFR	20	Red regular-range ASuW munitions for surface ships
JROFTP	20	Red ASW munitions for surface ships
JROLAA	20	Red air-to-air munitions for land-based aircraft
JROLAF	20	Red anti-surface munitions for land-based aircraft
JROSCM	20	Red attack munitions for missile-firing submarines
JROSTP	20	Red attack munitions for torpedo-firing submarines
JRSBAR	20	Red barrier submarines

(continued)

Table II-13. (Concluded)

Index Name	Code Number	Description
JRSCM	20	Red missile-firing submarines
JRSTP	20	Red torpedo-firing submarines
JSASCM	6	Red antiship cruise missile—ALCM plus SLCM
JSLBF	10	Red land-based bomber plus fighter a/c. Keyed to MYMRLA.
JSOFF	6	Red surface-launched ASuW ord.—long- plus regular-range
JTASCM	10	Red antiship cruise missile (ALCM or SLCM). Keyed to MZKRB1.
JTOFST	10	Red torpedo—surface- or submarine-launched. Keyed to MZKRS.
JXABSM	40	Numerical index for notional types of Red airbase def. SAMs
JXKBD	40	Numerical index for notional types of Blue land-bsd fighters
JXKBES	40	Numerical index for notional types of Blue escort surf.ships
JXKRB	40	Numerical index for notional types of Red bombers
JXKRSN	40	Numerical index for notional types of Red non-rsp. surf. shp
JXPPSM	40	Numerical index for notional types of Red power-proj-def SAM
JYKRSG	40	Numerical index for 4 notional subtypes of Red missile subs
MXABSM	2	Red land-based SAMs for airbase defense
MXKBD	2	Blue land-based fighter aircraft
MXKBES	2	Blue surface ships—escort ships
MXKRB	2	Red land-based aircraft—bomber
MXKRSN	2	Red surface ships—non-resupply
MXKRSS	6	Red surface ships—non-resupply plus resupply
MXLOC	12	Region
MXMPSG	12	Movement period for Red SAG
MXNMP	12	Movement period for (Blue) task force
MXPPSM	2	Red land-based SAMs for power-projection defense
MXSAG	12	Individual Red SAG
MXTABA	12	Number of capable Blue AAW ships
MXTABD	12	Number of capable Blue ASW ships
MXTABH	12	Number of Red missile hits
MXTABP	12	Number of Blue power-projection destruction units
MXTABS	12	Number of Red torpedo hits
MXTABV	12	Number of Red torpedo hits
MXTC	12	Blue SLOC-delivered cargo (NAVMOD SLOC model)
MY10	14	Numerical index: 1,...,10
MY11	14	Numerical index: 1,...,11
MY2	14	Numerical index—1 or 2
MY3	14	Numerical index—1, 2, or 3
MY5	14	Numerical index—1,...,5
MY6	14	Numerical index—1,...,6
MYCTFI	14	1—partial Red flight path; 2—full Red flight path
MYCTFM	14	1—Red bombers; 2—Red missiles
MYCV	14	Individual (Blue) aircraft carrier
MYIAB	14	Vulnerability of Red airbase—must be coded as 1 or 2
MYKBA	4	Blue sea-based attack or fighter aircraft
MYKBC	4	Blue land-attack cruise msl—submarine- or surface-launched
MYKRD	4	Red land-based fighter or interceptor aircraft
MYKRSB	4	Red submarines in regions—torpedo- plus missile-firing
MYKRSG	3	Red missile-firing submarines in regions
MYMBLA	14	1—attack mission; 2—escort mission (Blue land-based a/c)
MYMBSA	14	1—attack mission; 2—escort mission (Blue sea-based a/c)
MYMM	14	1—minimum; 2—maximum
MYMRLA	14	1—Red bombers (all types combined); 2—Red escort aircraft
MZIPR	16	Numerical index, but must be equal to MZKRA
MZKBA1	14	Blue power-projector—encompasses sea-based aircraft and CM
MZKBA2	6	Blue sea-based aircraft—attack, fighter, AEW, and ASW
MZKBD1	6	Blue land-based aircraft for ASuW—fighter and ASW aircraft
MZKBD2	6	Blue land-based aircraft for ASuW—fighter and ASW aircraft
MZKBS	6	Blue ship—surface ship plus direct-support submarine
MZKBSS	6	Blue surface ship—carrier, escort, and URG
MZKRA	6	Red land-based aircraft—bombers, fighters, and interceptors
MZKRB1	6	Red antiship missile platform—bomber or submarine
MZKRS	6	Red ship—surface ships (incl. resupply) + subs in regions
MZLOC1	16	Region+1
MZOBSF	6	Compound index encompassing many types of Blue ordnance
MZORSF	6	Compound index encompassing many types of Red ordnance

Table II-14. COLUMNS OF INGRES TABLE INDXMSG AND THEIR MEANINGS

Column Name	Type and Length	Meaning
indxnam	character 6	name of index
msg	character 60	associated descriptive message

C. VARIABLES

1. Introduction

The more disaggregated data input to the aggregator are organized into variables that correspond to the NAVMOD inputs. Each variable is identified by a variable name--a character string of (up to) 6 characters. For the most part, the name of each more disaggregated variable is the same as the name of some NAVMOD input, and the names of nearly all NAVMOD inputs are also used as the names of more disaggregated variables. A few exceptions exist--the names of some NAVMOD inputs are not used for more disaggregated variables, and the names of some more disaggregated variables are not used NAVMOD inputs. These exceptions are discussed in this section.

In this section, and in most of this documentation, the term "variable" is used to refer to a more disaggregated data variable--a portion of the more disaggregated data organized under a variable name. The terms "aggregator input" and "input used by the aggregator" will also be used for this.

a. Section Organization

The primary aim of this section is to explain the structure of the more disaggregated data, to aid the user in developing appropriate data values. The structure of a number of specific variables is discussed in detail.

The preprocessor makes use of several different categorizations of the variables. Variables in different groups are, in general, treated with different algorithms and often by different subroutines of the preprocessor. The appropriate taxonomies are discussed in Section C.2. As part of this discussion, the categorization of the input files of more disaggregated data is explained. Section C.3 discusses several specific groups of variables. A number of INGRES tables contain information about the variables or certain subsets of them, which are also described in the following paragraphs.

b. INGRES Table RVARINFO

Each variable is regarded as an array (scalars being considered as zero-dimensional arrays), and each dimension of this array has an associated index name. INGRES table RVARINFO (mnemonic for real variable information) contains the appropriate information on the dimensioning and index sequence of the variables used by the aggregator (including the NAVMOD inputs plus the additional variables that are entered as input to the aggregator but not to NAVMOD).[17] The structure of table RVARINFO is shown in Table II-15. Each variable has exactly one line of table RVARINFO devoted to it. The information in table RVARINFO must be preset before the aggregator is run; if a new variable is added to NAVMOD or to the preprocessor, the appropriate information must be entered in table RVARINFO.

The entry in column vcoder of table RVARINFO is a variable code number, much like the index code numbers used for indices. The variable code numbers form one taxonomy for variables.

Currently, a variable can have no more than a total of four dimensions (NAVMOD dimensions plus added dimensions). With simple formatting changes, this limit can be extended to six dimensions, and with straightforward programming changes, it could be extended beyond that. Currently, no NAVMOD input has more than three dimensions. The aggregator methodology cannot currently treat more than three notional dimensions, but could be extended to do so.

2. Taxonomies of Variables

The variables can be organized and categorized in a number of ways. All of these methods, described in the following paragraphs, are used somewhere in the aggregator and/or the auxiliary programs.

a. Variable Code Numbers

Each variable has an associated variable code number, listed in column vcoder of INGRES table RVARINFO. Table II-16 contains these code numbers and their meanings.

---

[17]The NAVMOD inputs not used by the aggregator have also been listed in table RVARINFO to facilitate presentation of information. The aggregator preprocessor does not access these rows of table RVARINFO.

Table II-15. COLUMNS OF INGRES TABLE RVARINFO AND THEIR MEANINGS

Column Name	Type and Length	Meaning
vnamer	character 6	name of variable
vcoder	integer 1	code number of variable, as shown in Table II-16, below
nodr	integer 1	number of dimensions of variable as it is input to the aggregator (0=scalar, 1=one-dimensional array, etc.). Encompasses NAVMOD indexing plus any added indices.
nodn	integer 1	number of dimensions of variable as it is used in NAVMOD
nfn	integer 1	2 for real variables, 1 for integer variables
indx1	character 6	name of index for first dimension (if any)
indx2	character 6	name of index for second dimension (if any)
indx3	character 6	name of index for third dimension (if any)
indx4	character 6	name of index for fourth dimension (if any)
indx5	character 6	not currently used--available for future use as name of index for fifth dimension (if any)
indx6	character 6	not currently used--available for future use as name of index for sixth dimension (if any)
fcati	integer 1	(not currently used)

Table II-16. CODE NUMBERS FOR AGGREGATOR VARIABLES, AS STORED IN INGRES TABLE RVARINFO

Code 1 -- NAVMOD effectiveness variables. All inputs to NAVMOD not mentioned below, including the ordnance capacities inputs listed in Reference [15].

Code 3 -- Resource inputs to NAVMOD, except for array RS. (See Table II-17).

Code 4 -- Limit variables that are inputs to NAVMOD, except for NKRS and MIMPSG. Variables: MIMP, NABSAM, NKBDPL, NKBES, NKRB, NLOC, NPPFFX, NPPSAM, NSAG, NTABA, NTABD, NTABH, NTABS, NTABV, NTC. For definitions, see Appendix F of Reference [2].

Code 5 -- Ordnance stock inputs to NAVMOD, except for arrays OVBTF and OVRRG. These inputs are listed, with definitions, in the ordnance stocks section of Reference [15].

Code 7 -- Inputs to NAVMOD that NAVMOD does not currently use. The aggregator does not accept data for these variables and does not attempt to produce aggregated values for them. They are listed in the "Inputs Not Currently Used" section of Reference [15].

Code 11 -- The allocation fraction variables, used by the aggregator but not used as inputs to NAVMOD itself. Variables: FABRSM, FATABT, FBESS, FPLBLB, FPPRSM, FRSFNR, FRSSG. For definitions, see Table II-20.

Code 13 -- Three array variables for Red submarines and ships: RSBTP, RSFNRP, and RSFRSP. These are used by the aggregator but not by NAVMOD itself. Instead, NAVMOD uses the array RS, and the aggregator code explicitly computes values for the appropriate elements of RS from RSBTP, RSFNRP, and RSFRSP. For definitions, see Table II-21.

Code 14 -- Two limit variables, NKRSN and NMPSG, used by the aggregator but not by NAVMOD itself (but the related variables NKRS and MIMPSG are inputs to NAVMOD). For definitions, see Table II-19.

Code 15 -- Special variables for certain reserve ordnance stocks. These variables are used by the aggregator but not by NAVMOD itself (instead, NAVMOD uses elements of the arrays OVBTF and OVRRG). For definitions, see the variables beginning with "OV" in Table II-21. (Aggregator code explicitly assigns aggregated values for the Code 15 variables to the corresponding elements of NAVMOD inputs OVBTF and OVRRG.)

Code 23 -- The NAVMOD input array RS. The aggregator uses the Code 13 variables RSBTP, RSFNRP, and RSFRSP and assigns aggregated values for these variables to the appropriate elements of RS.

Code 24 -- The NAVMOD input limit variables NKRS and (array) MIMPSG. The aggregator code explicitly computes values for these variables from the values of the Code 14 variables NKRSN and NMPSG.

Code 25 -- The NAVMOD input arrays OVBTF and OVRRG. The aggregator code explicitly computes values for appropriate elements of these arrays from the corresponding values of the Code 15 variables.

The vast majority of the variables are considered effectiveness variables, and have a variable code number of 1. Table II-17 contains the names of the variables that are not effectiveness variables. The name of each (more disaggregated) effectiveness variable is also the name of some NAVMOD input, and the effectiveness variables correspond to those NAVMOD inputs whose names do not appear in Table II-17.

Although the particular numerical values used for the different code numbers are essentially arbitrary, the computer programs do refer to them. Within the computer code of the aggregator itself, the only variable code number specifically mentioned is 1 (to distinguish effectiveness variables), but some of the auxiliary programs make use of the other code numbers.

Note that variables with codes 7, 23, 24, or 25 are NAVMOD inputs but not aggregator input variables, and variables with codes 11, 13, 14, or 15 are aggregator inputs but not NAVMOD inputs.[18] The variables with codes 13, 14, or 15 are associated with certain elements of certain NAVMOD input arrays.

b. Categorization of Effectiveness Variables

As mentioned in Chapter I, the more disaggregated data are stored on ASCII files. For each effectiveness variable, the file (file name and extension) that contains the more disaggregated data for that variable must be listed in column filname of INGRES table EVARINFO, in the row for that variable. Table EVARINFO (mnemonic for "extra variable information") contains certain information on the NAVMOD input variables (encompassing effectiveness variables, resource variables, ordnance stock variables, and input limit variables). Table II-18 shows the structure of table EVARINFO. Table EVARINFO is also used by the interaction selector preprocessor. Table EVARINFO has exactly one row for each NAVMOD input, and each effectiveness variable is also a NAVMOD input. The aggregator preprocessor and auxiliary programs ignore the rows of table EVARINFO that do not correspond to effectiveness variables.

Recall that the NAVMOD documentation has divided the NAVMOD input variables into a number of categories, based, for the most part, on the particular combat interaction that the variables are used to help model. These categories are not used by the NAVMOD

---

[18] For variables not used by NAVMOD, the value in column nodn of table RVARINFO is without meaning and is not accessed by the aggregator.

Table II-17. LISTING OF VARIABLES, OTHER THAN EFFECTIVENESS VARIABLES, ARRANGED BY CODE NUMBER

Code 3--resource variables input to both NAVMOD and the aggregator

ABRSAM	BSSNDS	RURGS	XASW
AESCAB	PLBLBD	SHEL	XASWLQ
AINTCP	PPRSAM	TOBBAR	XATTCK
ATABT	RSIBAR	TORBAR	XFGHTR
BESS	RSSG	XAEW	XPLAT
BSIBAR	RUNRCP	XAEWLQ	XURGS

Code 4--limit variables input to both NAVMOD and the aggregator

MIMP	NKRB	NSAG	NTABS
NABSAM	NLOC	NTABA	NTABV
NKBDPL	NPPFFX	NTABD	NTC
NKBES	NPPSAM	NTABH	

Code 5--ordnance stock variables input to NAVMOD and the aggregator (treated the same as resource variables)

ABANM	OIBFDS	OIRFAA	OIRSCM	OVRLAA
OIBAAA	OIBFFF	OIRFAS	OIRSTP	OVRLAF
OIBAAF	OIBFTP	OIRFDA	OVBBAR	PPANMS
OIBAAG	OIBSCM	OIRFDL	OVBLAA	
OIBAAS	OIBSTP	OIRFDS	OVBLAF	
OIBBAR	OIBUA	OIRFFL	OVBLAS	
OIBFCM	OIBUR	OIRFFR	OVBUL	
OIBFDA	OIRBAR	OIRFTP	OVRBAR	

Code 7--variables not currently used by NAVMOD and thus not input to the aggregator

FURGT0	IABAF	IPLADA	RSSPBS
FURGTP	IPIADA	IPPAF	VTSPBS
IAADA	IPICDA	PKSAAD	

Code 11--allocation fraction variables

FABRSM	FPPRSM
FATABT	FRSFNR
FBESS	FRSSG
FPLBLB	

Code 13--resource variables input to aggregator but not to NAVMOD

RSBTP	RSFNRP	RSFRSP

Code 14--limit variables input to aggregator but not to NAVMOD

NKRSN	NMPSG

(continued)

Table II-17. (Concluded)

Code 15--ordnance stock variables input to aggregator but not to NAVMOD

OVBAAA	OVBFDS	OVRFAA	OVRFFL
OVBAAF	OVBFFF	OVRFAS	OVRFFR
OVBAAG	OVBFTP	OVRFDA	OVRFTP
OVBAAS	OVBSCM	OVRFDL	OVRSCM
OVBFCM	OVBSTP	OVRFDS	OVRSTP
OVBFDA			

Code 23--resource variable input to NAVMOD but not to the aggregator
RS

Code 24--limit variables input to NAVMOD but not to the aggregator
MIMPSG NKRS

Code 25--ordnance stock variables input to NAVMOD but not to the aggregator
OVBTF OVRRG

Table II-18. COLUMNS OF INGRES TABLE EVARINFO AND THEIR MEANINGS

Column Name	Type and Length	Meaning
vname	character 6	name of variable
increm	integer 1	0 if the variable value can be replaced by Subroutine TIMET; 1 if the variable value can be incremented by Subroutine TIMET
cblock	character 6	name of COMMON block in which the variable is located in NAVMOD
rstcode	integer 1	restriction code (not used by aggregator)
rout	character 6	name of subroutine of NAVMOD in which the variable is used; MULTPL for variables used in more than one subroutine
catname	character 6	Category name for the variable. Category specification consists of category name and category number (column catno). Category name is often the same as the subroutine name in the rout column; MULTPL indicates variables belonging to more than one category; GENRAL indicates variables belonging to a general category (non-effectiveness variables)
catno	integer 1	category number for the variable
ndatlin	integer 4	number of lines of more disaggregated data for this variable (these data are stored in the file indicated in the filname column)
filname	character 45	name of file containing the more disaggregated data for this variable (file name and extension only; directory specification is in table PFNAMEF). This file is read into table RDATEFF,where its data are read by the aggregator.

model itself, but were introduced to help present the NAVMOD inputs in an organized fashion. These categories are explained in Appendix F of Reference [2], and Reference [15] presents definitions of the NAVMOD inputs, organized by category. The categories also appear in the INGRES table EVARINFO. As explained in References [2] and [15], each category is characterized by a category name (generally the name of a NAVMOD subroutine) and a category number, indicating combat interaction within a subroutine.

The different files of effectiveness data are not required to follow this NAVMOD categorization pattern. The auxiliary program GENEFFDAT, which generates the shell files for the more disaggregated effectiveness data, does make use of this categorization, however. With the exceptions noted below, Program GENEFFDAT generates a different shell file for the effectiveness variables in each given NAVMOD category. As explained in Chapter IV, Section C.5, the names of these files are based on the category names. Program GENEFFDAT stores these file names in (column filname of) INGRES table EVARINFO. If the user wants to rename or switch the effectiveness data files, the user can update column filname of table EVARINFO as appropriate, using QBF or SQL (References [12] and [6]). (The aggregator will then, of course, look to these new files for the more disaggregated data.)

With regard to the connection between the NAVMOD categorization of variables and the aggregator variable codes, the following points should be noted:

- The variables listed in the ordnance capacities category of Reference [15] are considered in the aggregator to be effectiveness variables.
- Conversely, the variables NPPFFX, NTABA, NTABD, NTABH, NTABS, and NTABV, which are in the NAVMOD categories POWERP.9, CTFMOD.11, CTFMOD.3, CTFMOD.12, CTFMOD.5, and MOVTF.4, respectively, are considered to be limit variables--not effectiveness variables--by the aggregator.
- Approximately 50 effectiveness variables fall into two or more categories. Program GENEFFDAT generates one shell file for the more disaggregated data for these variables (see Chapter IV, Section C.5).

c. Taxonomy by Input Files

As indicated in Chapter I, the aggregator has two parts. The first part computes values for the NAVMOD resource, ordnance, and limit variables and computes weights to aggregate the effectiveness variables. The second part performs the weighted averaging, computing values for the NAVMOD effectiveness variables.

The more disaggregated data read in by the first part must all be contained in one file. (The name of this file is specified by the user in the input options guide file.) The variables accessed from this file are those with variable codes 3, 4, 5, 11, 13, 14, and 15, encompassing the more disaggregated resource, ordnance, and limit variables, and the allocation fraction variables.

The more disaggregated data read in by the second part of the aggregator are all code-1 (effectiveness) variables. They are separated according to the ASCII files specified in INGRES table EVARINFO. The second part of the aggregator processes effectiveness variables only, and no effectiveness variable is processed by the first part.

d. Taxonomy by Algorithmic Treatment

The first part of the aggregator processes the following types of variables:

- limit variables (codes 4 and 14);
- resource and ordnance variables (codes 3, 5, 13, and 15); and
- allocation fraction variables (code 11).

These three classes of variables are handled by different algorithms in the program. Limit variables are treated by Subroutine LMVSET, as described in Chapter III, Section C. Resource and ordnance variables are processed by Subroutine RSOWGT, as described in Chapter III, Section E. Allocation fractions are also processed by Subroutine RSOWGT, but are processed differently from the resource and ordnance variables (see Chapter III, Section E).

Special computer code exists to transform the code-13, code-14, and code-15 variable values into values for NAVMOD variables. These corresponding NAVMOD variables--RS, OVBTF, OVRRG, NKRS, and MIMPSG (the code-23, 24, and 25 variables)--are referred to by name in the computer code. This code would probably have to be altered if the aggregator methodology were to be transferred to other models.

3. Discussion of Specific Types of Variables

a. Integer Variables--A Special Note

Some of the inputs to NAVMOD are integer-valued variables. NAVMOD follows the standard FORTRAN conventions for naming variables--variables with names beginning with the letters I through N are integer; the others are real. (NAVMOD contains no character input variables.) The limit variables and approximately 50 of the (input) effectiveness variables are integer. The information on variable type is stored in the information file read by the aggregator (and also in INGRES table RVARINFO).

All more disaggregated data values should be entered with a decimal point, and all weighted averaging is performed in real (double precision) arithmetic. When the aggregated value has been computed and is about to be stored (for later output on the file of

aggregated values), Subroutine PUTONE checks the variable type; if the type is integer, the value stored is computed as the integer part of (.001 plus the provisional aggregated value).

In effect, this algorithm leads to simple copying of integer data values. The integer inputs are mostly scalars, and the few integer input arrays have numerical indices. Unless the user wishes (for some reason) to add indices to integer variables, each weighted averaging procedure will degenerate into a "weighted average" of one term, with a weight of 1.0. Thus, an integer value for an integer-valued NAVMOD effectiveness variable that is specified (with a decimal point) in the more disaggregated data will in effect be transferred as is to the aggregated data.[19]

The user values for the limit variables should be (positive) integer values but should be specified with a decimal point in the more disaggregated data. The weighted averaging procedure for effectiveness variables is not performed on limit variables. but the truncation procedure of Subroutine PUTONE is applied to them (to avoid roundoff error).

b. Limit Variables

Limit variables indicate the number of elements of an array dimension that the NAVMOD code will actually use, and frequently indicate the number of notional types of a resource to be played, for those resources of which NAVMOD can model multiple types.

Table II-19 lists and defines the limit variables that are part of the more disaggregated data. Except for NKRSN and NMPSG, these limit variables are also inputs to NAVMOD. Appendix F of Reference [2] gives more complete definitions of these variables, focusing on their use in NAVMOD. As indicated previously, the NAVMOD input limit variables NKRS and MIMPSG are not, in themselves, in the more disaggregated data; NKRSN and NMPSG substitute for them.

The user should decide the values of all limit variables prior to running the aggregator preprocessor and these values should not be changed in the data for NAVMOD itself.[20] All limit variables should have positive integer values, with the possible exception of NSAG and NTC. A zero value for a limit variable can result in data associated with that variable not being aggregated.

---

[19]The variable IPRSRA may need special handling by the user. See the discussion in Chapter IV, Section B.3.c.2). Also, see Chapter IV, Section B.2.c.

[20] Some of the limit variables can later be changed. See the discussion of MIMPSG, in this section, and the comments in Chapter IV, Section B.2.f.

Table II-19. LIMIT VARIABLES INPUT TO THE AGGREGATOR

MIMP—Maximum number of movement periods for the task force.

NABSAM—Number of types of Red SAMs defending vulnerable Red a rbases.

NKBDPL—Number of types of Blue land-based fighter aircraft.

NKBES—The number of types of Blue escort (surface) ships in the (Blue) task force.

NKRB—Number of types of Red bombers.

NKRSN—Number of notional types of Red surface ships, not including resupply ships. NAVMOD input NKRS is set to NKRSN+3, encompassing NKRSN plus one notional type of resupply ship plus two notional types of Red submarines in regions.

NLOC—Number of possible regions (excluding region zero) in which the task force can be located.

NMPSG—Maximum number of movement periods for any Red SAG. NAVMOD input MIMPSG(ISAG) is set to NMPSG for all ISAG. Reset it later if desired.

NPPFFX—Number of bending points of the input piecewise linear function PPCVFX(I), which is used to compute the value of pcwer projection sorties flown against fixed targets.

NPPSAM—Number of types of Red SAMs involved in Blue power projection interactions.

NSAG—Number of SAGs (surface action groups) the Red surface ships are organized into. (This input can be zero.)

NTABA—Number of rows of the input array TAB10T that will be used by the code (of Subroutine CTFMOD).

NTABD—Number of entries of the input vector PDINV that will be used by the code (of Subroutine CTFMOD).

NTABH—Number of rows of the input array TAB13T that will be used by the code (of Subroutine CTFMOD).

NTABS—Number of entries of the input vector TAB12 that will be used by the code (of Subroutine CTFMOD).

NTABV—Number of entries of the input vector VTTAB that will be used by the code (of Subroutine MOVTF).

NTC—Number of types of cargo carried by Blue ships (used in SLOC model). This variable can be zero (and is ignored) if ISLOC = 0.

The input to NAVMOD NKRS represents the number of types of Red ships modeled, including resupply ships and non-barrier submarines (but not including barrier submarines and barrier tender ships). (Note that reviewing the discussion in Reference [2] of NAVMOD's modeling of Red resupply ships may be helpful.) NAVMOD models one type of Red (non-barrier) resupply ship, one type of Red non-barrier torpedo-firing submarine, one type of Red non-barrier missile-firing submarine[21], and a user-controlled number of types of Red non-resupply surface ships (excluding barrier tenders). The input to the aggregator NKRSN represents the number of types of these Red non-resupply surface ships the user wishes to model in the NAVMOD data. (An allocation fraction operates in conjunction with this limit variable, as explained in the next section.) In the aggregator code, the value for NKRS is computed (and later generated as output) as 3 plus the value input for NKRSN, encompassing the total number of notional resource types in the preceding four resource categories.

In the resultant aggregated data for NAVMOD, dimensions on notional type of Red ship follow the pattern:

- element 1 corresponds to Red torpedo-firing submarines;
- element 2 corresponds to Red missile-firing submarines (aggregation of all subtypes);
- elements 3 through NKRS–1 correspond to the NKRSN types of Red surface ships, not including resupply ships; and
- element NKRS corresponds to Red resupply ships;

Dimensions on notional type of Red *surface ship* follow the pattern:

- elements 1 through NKRS–3 correspond to the NKRSN types of Red surface ships, not including resupply ships;
- element NKRS–2 corresponds to Red resupply ships.

The number of Red resupply ships, denoted in the NAVMOD inputs by RS(NKRS), can be set to zero in NAVMOD if the user does not want to model Red resupply ships.

The limit variable MIMPSG, and its analog in the more disaggregated data, NMPSG, apply to NAVMOD's modeling of Red surface ships in SAGs. (The reader

[21] Four different subtypes of such submarines are modeled, as explained in Chapter II, Section B.5, of Reference [1].

might wish to review the discussion of this modeling in Chapter III, Section B.1, of Reference [1].) MIMPSG is an array: element MIMPSG(ISAG) is the number of movement periods for the ISAGth Red SAG, where ISAG varies from 1 through the input limit variable NSAG. Since MIMPSG is the only non-scalar NAVMOD limit variable, it was decided to use a special purpose algorithm rather than to develop additional methodology to handle a nested limit variable structure. Accordingly, the (scalar) aggregator input NMPSG represents the number of movement periods for a typical Red SAG, and the value read in for NMPSG is assigned to the NAVMOD input array element MIMPSG(ISAG), for each value of ISAG from 1 through the full dimension limit (which is accessible to the aggregator). The user can then edit the file of aggregated data to change the values of MIMPSG as desired. Note that the only NAVMOD inputs that use the elements of MIMPSG as limit variables are the arrays LGSGMP and LSAGMP; these should also be edited as appropriate.

The aggregator methodology cannot currently handle, in a general fashion, limit variables in the aggregated data that span two or more resource categories and/or are themselves arrays. Each limit variable currently treatable in the more disaggregated data is a scalar and is associated with either exactly one resource category or a numerical index (with index code 12). The NAVMOD limit variables that do not fit this pattern have had special hard-coded procedures written for them. Improving the aggregator methodology to handle additional patterns in the limit variables is an area for future work.

The limit variables are listed in INGRES table RVARINFO, but the aggregator preprocessor does not access them from table RVARINFO (some of the auxiliary programs do). Instead, the preprocessor accesses them from table LIMVINFO (mnemonic for limit variable information), in Subroutine LMVSET. This table is discussed in full in Chapter III, Section C. It contains a row for each (code-4 and code-14) limit variable.

c. Allocation Fraction Variables

In the aggregator, each resource category has an associated list of actual resource types, listed in INGRES table GENTYP0. For some resources in NAVMOD, several notional types or subtypes can be modeled. Usually, this number is an input, indicated by the value of a limit variable.[22] Examples are Blue escort ships, Blue land-based fighter

[22]The methodology of allocation fraction variables is also applicable to resources for which a fixed number of notional subtypes is modeled. (In NAVMOD, the only resource so modeled is Red missile-firing submarines.) It is also applicable to resources for which a fixed number (greater than unity) of notional

aircraft, and Red bombers. For such resources, the allocation fraction variables indicate the correspondence between the actual resource types and the different notional types. The allocation fraction variables are not inputs to NAVMOD itself, but are inputs to the aggregator preprocessor. The data values for them should be entered in the same file as the more disaggregated resource, ordnance, and limit variable data. (The auxiliary program GENROLDAT generates a shell file for these data, as described in Chapter IV, Section C.6.)

Several slight variations and subcases exist in the aggregator's treatment of allocation fraction variables, but the basic concept is as follows. An allocation fraction variable is associated with each resource category whose corresponding NAVMOD resource can have multiple types or subtypes. The interpretation of the variable is that f_{ij} represents the proportion of type-i actual resource that is to be considered as type-j notional resource. The index i varies over the list of actual types in table GENTYP0 (for the appropriate resource category), and the index j varies from 1 through the number of notional types to be simulated, which is either fixed or equal to the value of the limit variable associated with the resource category. (This value must be determined before the aggregator is run. This value is read in by the aggregator and stored in column ivalntl of INGRES table INDXBINFO before the aggregator accesses the values of the allocation fraction variables.) It is required that for each value of i,

$$\sum_j f_{ij} = 1$$

indicating that all actual resources are to be considered as some notional resource type. Let A_i denote the number of (units of) actual resource type i (in the resource category under consideration), where i ranges as described above. (This number is the value of some more disaggregated resource variable and is accessed by Subroutine RSOWGT.) For each value of j from 1 through the associated upper limit, the number of units of notional resource type j is computed as

$$B_j = \sum_i f_{ij} A_i.$$

This value is stored and is later generated as output on the file of aggregated resource,

---

types is modeled. NAVMOD does not (currently) have any of these latter resources, whose associated resource categories would be characterized by code 3 in INGRES table GENUSINFO.

ordnance, and limit variables. (For a more detailed explanation, see the description of Subroutine RSOWGT in Chapter III, Section E.[23]) (For maximum consistency of results, the A_i and f_{ij} should be chosen such that each B_j will be nonzero.)

The allocation fraction variables can be used to group actual resources, as follows. If actual resource type i is to be considered as notional resource type j, set $f_{ij} = 1$ and set $f_{ij'} = 0$ for that i and all other j'. The other actual resource types i' to be considered as notional resource type j would be similarly treated. This use of the allocation fraction variables might frequently occur, as it might often be the case that the number of types of some resource the user wants NAVMOD to model is somewhat less than the number of actual types of that resource that are to be considered in the aggregator. (For example, a typical NAVMOD run might consider 4 or 5 different notional types of Blue escort ships, but the user may wish to use more disaggregated data for 10 or 20 actual types of Blue escort ships. The allocation fractions allow these actual types to be grouped into notional types as desired.)

The allocation fraction variables are listed, with definitions, in Table II-20. The indices JXABSM, JXKRB, etc., appearing in these definitions are code-40 indices, and are used by the auxiliary program GENROLDAT. Each of these indices is listed in INGRES table INDXBINFO; in its row, the corresponding code-2 index (MXABSM, MXKRB, etc.), appears in column indxrnam. (If changes to NAVMOD make it appropriate to add new allocation fraction variables to the aggregator, these variables will need to be entered in table RVARINFO with variable code 11 and indexing similar to the current allocation fraction variables. The indices for the variables will need to be entered in table INDXBINFO, if they are not currently there.)

d. Resource Variables

The more disaggregated resource variables indicate amounts (numbers of units) of actual resource types. (Ordnance stocks are treated by the same methodologies as other resources, and the term "resource" should be thought of as encompassing ordnance.) From the more disaggregated resource variables are computed two groups of quantities:

---

[23]The allocation fractions are listed in INGRES table RVARINFO (with variable code 11) and are accessed there by auxiliary programs. In the aggregator itself, however, the names of the allocation fraction variables are accessed (by Subroutine RSOWGT) from INGRES table RESGENUS, along with the names of the corresponding resource categories and resource variables. See Chapter III, Section E.

Table II-20. ALLOCATION FRACTION VARIABLES

FABRSM(MXABSM,JXABSM)—Fraction of actual-type-MXABSM Red airbase defense SAMs that will be considered to be notional-type-JXABSM such SAMs.
MXABSM ranges over the actual types in the resource category RMABD.
The numerical index JXABSM ranges from 1 to the limit NABSAM.

FATABT(MXKRB,JXKRB)—Fraction of actual-type-MXKRB Red bombers that will be considered to be notional-type-JXKRB such bombers.
MXKRB ranges over the actual types in the resource category RLBMR.
The numerical index JXKRB ranges from 1 to the limit NKRB.

FBESS(MXKBES,JXKBES)—Fraction of actual-type-MXKBES Blue escort ships that will be considered to be notional-type-JXKBES such ships.
MXKBES ranges over the actual types in the resource category BFESC.
The numerical index JXKBES ranges from 1 to the limit NKBES.

FPLBLB(MXKBD,JXKBD)—Fraction of actual-type-MXKBD Blue land-based fighter aircraft that will be considered to be notional-type-JXKBD such fighters.
MXKBD ranges over the actual types in the resource category BLFTR.
The numerical index JXKBD ranges from 1 to the limit NKBDPL.

FPPRSM(MXPPSM,JXPPSM)—Fraction of actual-type-MXPPSM Red power-projection-defense SAMs that will be considered to be notional-type-JXPPSM such SAMs.
MXPPSM ranges over the actual types in the resource category RMPPD.
The numerical index JXPPSM ranges from 1 to the limit NPPSAM.

FRSFNR(MXKRSN,JXKRSN)—Fraction of actual-type-MXKRSN Red surface ships, not including resupply ships, that will be considered to be notional-type-JXKRSN such ships
MXKRSN ranges over the actual types in the resource category RFNRP.
The numerical index JXKRSN ranges from 1 to the limit NKRSN.
(This limit itself is a new input variable, as indicated earlier.)

FRSSG(MYKRSG,JYKRSG)—Fraction of actual-type-MYKRSG Red missile-firing submarines in regions that will be considered to be (sub)type-JYKRSG such submarines.
MYKRSG ranges over the actual types in the resource category RSCM.
The numerical index JYKRSG ranges from 1 to 4, covering the four subtypes of Red missile-firing submarines.

aggregated values suitable for the resource variables used by NAVMOD; and, perhaps more important, weights that will be used in computing the aggregated effectiveness variable values. The aggregated values are output on the file RESORDLM.AGD, which also contains the data for the NAVMOD limit variables. The weights are stored in the INGRES table GENUSTYWGT and specify the relative composition of each notional resource type in terms of actual resource types.

The aggregated values and weights are computed by Subroutine RSOWGT of the aggregator, and the relevant algorithms are described in detail in Chapter III, Section E.

Most of the more disaggregated resource variables have the same names as NAVMOD resource variables. (The exceptions are detailed in the following paragraphs.) These variable names are presented in Table II-17, under variable codes 3 and 5. If the corresponding NAVMOD resource variable is d-dimensional (for scalars, interpret d as zero), then the first d indices of the more disaggregated resource variable correspond to those of the NAVMOD variable. For most of the variables for resources of which NAVMOD models only one type, an index on actual resource type has been added (so that the more disaggregated resource variable is (d + 1)-dimensional).[24] This added index structure should not be changed by the user (unlike the case of effectiveness variables, where the user has considerable freedom to specify added indices).

In the more disaggregated data, as in NAVMOD, many of the resource variables have a dimension on region. That is, a separate amount of resource must be specified for each region, for each actual resource type. (The number of regions must be determined by the user and specified by the limit variable NLOC.) For each region, amounts of notional resources for that region are computed from the amounts of actual resources for that region. For computing the weights, sums over different regions are used, as indicated in Chapter III, Section E. Some Red resources have a dimension on Red airbase type (1--vulnerable to Blue sea-based air; 2--invulnerable). The treatment of Red airbase types is similar to the case of regions, except the number of airbase types is fixed, not user-specified.

---

[24]The scalar NAVMOD resource variables SHEL and RUNRCP, which represent the number of Red aircraft shelters on vulnerable Red airbases and the amount of runway repair material on vulnerable Red airbases, respectively, remain scalars in the more disaggregated data, and values input for them are simply copied into the aggregated data. Resource categories are not associated with these resources--i.e., the aggregator does not (currently) consider different actual types of shelters and runway repair material. If appropriate, it could be modified to dc ,`.

In some cases, the same type of actual resource might be associated with two different resource categories. For instance, the F/A-18 might be listed in both the Blue sea-based attack aircraft and Blue sea-based fighter aircraft categories. In such cases, two different sets of more disaggregated data exist for that resource, and these sets are treated independently[25] In the F/A-18 case, the more disaggregated data entry with variable name XATTCK and component name F/A-18 would indicate the number of F/A-18s to be treated as attack aircraft, and the entry with variable name XFGHTR and component name F/A-18 would indicate the number of F/A-18s to be treated as fighter aircraft. This structure is consistent with a pre-allocation of the F/A-18 stock between these two resource categories.

The NAVMOD (input) resource variables RS, OVBTF, and OVRRG have the property that not all elements of these arrays correspond to resources from the same resource category. Some elements of RS indicate numbers of Red torpedo-firing submarines, other elements indicate numbers of Red non-resupply ships, and other elements indicate numbers of Red resupply ships. The different elements of OVBTF represent reserve stocks of the different types of ordnance used by the Blue task force. NAVMOD models the various types of ordnance in different ways. Accordingly, the aggregator considers each type to be a separate resource category. Similarly, the different elements of OVRRG represent reserve stocks of the different types of ordnance used by Red non-barrier ships and submarines. (Information on the meanings of the specific elements of OVBTF and OVRRG appears in Tables II-1 and II-2 of Reference [2].)

Currently, the algorithms of the preprocessor require that each resource variable name in the more disaggregated data be associated with exactly one resource category. The program could be improved to be more flexible in this regard, but for the present time, the following procedure has been used to deal with the NAVMOD inputs RS, OVBTF, and OVRRG. Several more disaggregated resource variable names have been invented and have been given variable code number 13 or 15, as indicated in Table II-17; they are defined in Table II-21. Each variable is associated with one resource category and with the

[25]From an algorithmic standpoint, this case is different from the case of multiple notional types within the same resource category, in which an allocation fraction variable is used to apportion one specified stock of actual resource among various notional types. Although this difference arises artificially, out of the conventions by which NAVMOD models different kinds of resources, it does affect the structure of the inputs.

Table II-21. NEW ORDNANCE STOCK AND RESOURCE VARIABLES

Note: The variables below that begin with "OVB" are the more disaggregated analogues of the corresponding components of the NAVMOD input OVBTF. The variables below that begin with "OVR" are the more disaggregated analogues of the corresponding components of the NAVMOD input OVRRG.

OVBAAA(JBOAAA)—Reserve inventory of Blue air-to-air missiles for Blue sea-based aircraft.
JBOAAA ranges over actual ordnance types in category BOAAA.

OVBAAF(JBOAAF)—Reserve inventory of Blue anti-surface missiles for Blue sea-based aircraft.
JBOAAF ranges over actual ordnance types in category BOAAF.

OVBAAG(JBOAAG)—Reserve inventory of Blue air-to-ground ordnance (e.g., bombs) for Blue sea-based aircraft.
JBOAAG ranges over actual ordnance types in category BOAAG.

OVBAAS(JBOAAS)—Reserve inventory of Blue air-dropped ASW ordnance (torpedoes) for Blue sea-based ASW aircraft and helicopters.
JBOAAS ranges over actual ordnance types in category BOAAS.

OVBFCM(JBOFCM)—Reserve inventory of Blue surface-launched ground-attack cruise missiles.
JBOFCM ranges over actual ordnance types in category BOFCM.

OVBFDA(JBOFDA)—Reserve inventory of Blue AAD missiles.
JBOFDA ranges over actual ordnance types in category BOFDA.

OVBFDS(JBOFDS)—Reserve inventory of Blue SSD missiles.
JBOFDS ranges over actual ordnance types in category BOFDS.

OVBFFF(JBOFFF)—Reserve inventory of Blue surface-launched anti-surface-ship missiles.
JBOFFF ranges over actual ordnance types in category BOFFF.

OVBFTP(JBOFTP)—Reserve inventory of Blue surface-launched torpedoes (or other ASW ordnance).
JBOFTP ranges over actual ordnance types in category BOFTP

OVBSCM(JBOSCM)—Reserve inventory of Blue submarine-launched ground-attack cruise missiles.
JBOSCM ranges over actual ordnance types in category BOSCM.

OVBSTP(JBOSTP)—Reserve inventory of Blue submarine-launched torpedoes (or other ASW ordnance).
JBOSTP ranges over actual ordnance types in category BOSTP

OVRFAA(JROFAA)—Reserve inventory of Red air-to-air missiles for Red sea-based aircraft.
JROFAA ranges over actual ordnance types in category ROFAA

OVRFAS(JROFAS)—Reserve inventory of Red air-dropped ASW ordnance (torpedoes) (Air platform is modeled implicitly.)
JROFAS ranges over actual ordnance types in category ROFAS.

(continued)

Table II-21. (Concluded)

OVRFDA(JROFDA)—Reserve inventory of Red AAD missiles.
JROFDA ranges over actual ordnance types in category ROFDA.

OVRFDL(JROFDL)—Reserve inventory of Red long-range shipboard SAMs.
JROFDL ranges over actual ordnance types in category ROFDL.

OVRFDS(JROFDS)—Reserve inventory of Red SSD missiles.
JROFDS ranges over actual ordnance types in category ROFDS.

OVRFFL(JROFFL)—Reserve inventory of Red long-range antiship missiles.
JROFFL ranges over actual ordnance types in category ROFFL.

OVRFFR(JROFFR)—Reserve inventory of Red regular-range antiship missiles.
JROFFR ranges over actual ordnance types in category ROFFR.

OVRFTP(JROFTP)—Reserve inventory of Red surface-launched torpedoes (or other ASW ordnance).
JROFTP ranges over actual ordnance types in category ROFTP.

OVRSCM(JROSCM)—Reserve inventory of Red submarine-launched antiship cruise missiles.
JROSCM ranges over actual ordnance types in category ROSCM.

OVRSTP(JROSTP)—Reserve inventory of Red submarine-launched torpedoes.
JROSTP ranges over actual ordnance types in category ROSTP.

RSBTP(JRSTP,L)—Number of Red torpedo-firing submarines of type JRSTP in region L. This is the more disaggregated analogue of the NAVMOD input RS(.,L) where the first component is equal to 1.
JRSTP ranges over actual type of submarine, in resource category RSTP.
L=1,...,NLOC.

RSFNRP(JRFNRP,L)—Number of Red surface ships, not including resupply ships, of type JRFNRP in region L. This is the more disaggregated analogue of the NAVMOD input RS(.,L) for values of the first component between 3 and NKRS-1. (NKRS is an input limit variable for NAVMOD that indicates the number of types of Red ships; it is equal to NKRSN+3, where NKRSN is an input limit variable for the aggregator.)
JRFNRP ranges over actual type of (non-resupply) surface ship, in resource category RFNRP.
L=1,...,NLOC.

RSFRSP(JRFRSP,L)—Number of Red (surface) resupply ships of type JRFRSP in region L. This is the more disaggregated analogue of the NAVMOD input RS(.,L) for values of the first component equal to NKRS. (NKRS is as described above.)
JRFRSP ranges over actual type of resupply ship, in resource category RFRSP.
L=1,...,NLOC.

subset of the elements of RS, OVBTF, or OVRRG that corresponds to NAVMOD values for resources in that category. More disaggregated data lines are associated with these invented variables (and appear in the shell file), and the user should supply appropriate data values for them. From these values, Subroutine RSOWGT computes aggregated resource values via one of its aggregation algorithms, and then assigns the result to the appropriate element of RS, OVBTF, or OVRRG. The names RS, OVBTF, and OVRRG are hard-coded in the computer program. If the aggregator program is adapted to other models, this will probably require changes. (As with other resources, Subroutine RSOWGT also computes weights for the actual resource types composing each notional resource type and stores them in table GENUSTYWGT.)

Whether identical in name to a NAVMOD variable or not, the more disaggregated resource variables fall into nine different indexing patterns. These patterns, and the corresponding resources and variables, are shown in Table II-22. Each pattern has been assigned a code number. Each pattern is linked to a different algorithm (aggregation method) for computing the aggregated resource values and weights. The algorithms are applied by Subroutine RSOWGT of the aggregator and are explained in detail in Chapter III, Section E. For each resource, the resource category, resource variable name(s), and pattern code number are stored in INGRES table RESGENUS. The column structure of table RESGENUS is presented in Section A.3, and its contents, which are fixed by the structure of NAVMOD and should be changed only if NAVMOD changes, are discussed in the explanation of Subroutine RSOWGT, in Chapter III, Section E.

D. REMARKS ON THE MORE DISAGGREGATED DATA

Much of the information in this section is contained in preceding sections; it is restated here to summarize how the structures and concepts included in this chapter relate to the form of the more disaggregated data.

Each variable (in the more disaggregated data) is associated with a sequence of indices, one for each dimension (notional or added) of the variable. Each index is associated with a set of component names. Thus a sequence of sets of component names is associated with each variable, one set for each index associated with the variable. The number of sets, denoted here by D, equals the total number of dimensions of the variable. The cartesian product of these sets is a collection of D-tuples of component names, and each such D-tuple is associated with a more disaggregated data value. A more

Table II-22. INDEXING PATTERNS FOR RESOURCE VARIABLES

Pattern 1—NAVMOD resource variable is scalar; more disaggregated variable adds dimension on actual resource type

BSSNDS—Blue direct-support submarines
OIBUA—Blue sonobuoys for sea-based patrol ASW aircraft
OIBUR—Blue sonobuoys for reactive (sea-based) ASW helicopters
OVBLAA—Blue air-to-air munitions for land-based aircraft
OVBLAF—Blue anti-surface munitions for land-based aircraft
OVBLAS—Blue ASW munitions for land-based aircraft
OVBUL—Blue sonobuoys for land-based patrol ASW aircraft
OVRLAA—Red air-to-air munitions for land-based aircraft
OVRLAF—Red anti-surface munitions for land-based aircraft
XAEW—Blue sea-based AEW aircraft
XASW—Blue sea-based ASW aircraft
XATTCK—Blue sea-based attack aircraft
XFGHTR—Blue sea-based fighter aircraft
XPLAT—Blue surface ships—carriers

Pattern 2—NAVMOD resource variable (if such exists) has dimension on region; more disaggregated variable adds dimension on actual resource type

BSIBAR—Blue barrier submarines
RSBTP—Red torpedo-firing submarines
RSFRSP—Red surface ships—resupply ships
RSIBAR—Red barrier submarines
TOBBAR—Blue surface ships—barrier tenders
TORBAR—Red surface ships—barrier tenders
XAEWLQ—Blue land-based AEW aircraft
XASWLQ—Blue land-based ASW/ASuW aircraft

Pattern 3—NAVMOD resource variable has dimension on Red airbase type (vulnerable or invulnerable); more disaggregated variable adds dimension on actual resource type

AESCAB—Red land-based aircraft—fighter/escort
AINTCP—Red land-based aircraft—interceptor

Pattern 4—NAVMOD resource variable has dimension on notional resource type; more disaggregated variable substitutes actual resource types for notional; allocation fraction variable is used

ABANM—Red missiles for airbase defense land-based SAMs
ABRSAM—Red land-based SAMs for airbase defense
BESS—Blue surface ships—escort ships
PPANMS—Red missiles for power-projection-defense land-based SAMs
PPRSAM—Red land-based SAMs for power-projection defense

(continued)

Table II-22. (concluded)

Pattern 5—NAVMOD resource variable (if such exists) has dimensions on notional resource type and region; more disaggregated variable substitutes actual resource types for notional and keeps region dimension; allocation fraction variable is used

PLBLBD—Blue land-based fighter aircraft
RSFNRP—Red surface ships—non-resupply
RSSG—Red missile-firing submarines

Pattern 6—NAVMOD resource variable has dimensions on Red airbase type and notional resource type; more disaggregated variable substitutes actual resource types for notional and keeps airbase type dimension; allocation fraction variable is used

ATABT—Red land-based aircraft—bomber

Pattern 7—Two NAVMOD resource variables, for active and reserve stocks; more disaggregated variables add dimension on actual resource type (active stock variable shown)

OIBAAA—Blue air-to-air munitions for sea-based aircraft
OIBAAF—Blue anti-surface munitions for sea-based aircraft
OIBAAG—Blue air-to-ground munitions for sea-based aircraft
OIBAAS—Blue ASW munitions for sea-based aircraft
OIBFCM—Blue land-attack cruise missiles for surface ships
OIBFDA—Blue AAD munitions for surface ships
OIBFDS—Blue SSD munitions for surface ships
OIBFFF—Blue anti-surface munitions for surface ships
OIBFTP—Blue ASW munitions for surface ships
OIBSCM—Blue land-attack cruise missiles for direct-support submarines
OIBSTP—Blue ASW/ASuW munitions for direct-support submarines
XURGS—Blue surface ships—URG ships

Pattern 8—Two NAVMOD resource variables, for active and reserve stocks; active stock variable has dimension on region; more disaggregated variables add dimension on actual resource type (active stock variable shown)

OIBBAR—Blue munitions for barrier submarines
OIRBAR—Red munitions for barrier submarines
OIRFAA—Red air-to-air munitions for sea-based aircraft
OIRFAS—Red air-dropped ASW munitions for surface ships
OIRFDA—Red AAD munitions for surface ships
OIRFDL—Red long-range SAMs for surface ships
OIRFDS—Red SSD munitions for surface ships
OIRFFL—Red long-range ASuW munitions for surface ships
OIRFFR—Red regular-range ASuW munitions for surface ships
OIRFTP—Red ASW munitions for surface ships
OIRSCM—Red attack munitions for missile-firing submarines
OIRSTP—Red attack munitions for torpedo-firing submarines

Pattern 9—NAVMOD resource variable is scalar, same as more disaggregated variable; simple copy of value

RUNRCP—Red runway repair material on vulnerable airbases
SHEL—Red aircraft shelters on vulnerable airbases

disaggregated data element can be considered to consist of a variable name, a D-tuple of component names, and a data value.

The more disaggregated data appear on a series of ASCII files; there is one line for each more disaggregated data element. Table II-23 shows the column structure that these files must follow. This column structure applies to all types of variables in the more disaggregated data: effectiveness, resource, limit, and allocation fraction variables. The auxiliary programs GENEFFDAT and GENROLDAT, which generate shell files for the more disaggregated data, have the appropriate variable name and component name information in the columns indicated in Table II-23, and 0.00000 in columns 64 through 70. The desired data value can be entered with a text editor. (Chapter I, Section B.1, contains some examples of portions of more disaggregated data files.)

The more disaggregated data for resource, ordnance, limit, and allocation fraction variables are all located in the same file. This file is read into INGRES table RDATROL, where the appropriate data are accessed by the aggregator program. The column structure of table RDATROL is shown in Table II-24. To avoid unduly slow speed, table RDATROL is modified to an ISAM storage structure on the columns vname, comp1, comp2, comp3, and comp4, via a MODIFY command embedded in the code. This necessitates some restrictions on who can run the program, as discussed in Chapter IV, Section A.

The more disaggregated data for effectiveness variables can be located in a collection of different files, as explained in Section C.2.b. Some or all of these files are read into INGRES table RDATEFF, where the appropriate data are accessed by (Subroutine AGGEFF of) the aggregator program.[26] The column structure of table RDATEFF is identical to that of table RDATROL, as shown in Table II-24. To avoid unduly slow speed, table RDATEFF is modified to an ISAM storage structure on the columns vname, comp1, comp2, comp3, and comp4, each time new data is read into it. Again, this necessitates some restrictions on who can run the program.

The total number of dimensions (notional plus added) of a variable cannot exceed 4. Changing the format of the files of more disaggregated data would allow this limit to be

[26]As discussed in the documentation of Subroutine EFFVAR, in Chapter III, Section G, the user can control which effectiveness data files are read in.

Table II-23. COLUMN STRUCTURE OF MORE DISAGGREGATED DATA FILES

Columns	Contents
1-6	Variable name
7	Blank
8-17	First component name (if any)
18	Blank
19-28	Second component name (if any)
29	Blank
30-39	Third component name (if any)
40	Blank
41-50	Fourth component name (if any)
51-56	Blank
57-70	Data value (use decimal point)[a]

[a]See the discussion of integer variables in Section C.3.a. Values for integer variables should still be entered with a decimal point.

Table II-24. COLUMNS OF INGRES TABLES RDATROL AND RDATEFF AND THEIR MEANINGS

Column Name	Type and Length	Meaning
vnamer	character 6	name of input variable
fcati	integer 1	(not currently used)
comp1	character 10	actual weapon type or component name in component 1 (if any)
comp2	character 10	actual weapon type or component name in component 2 (if any)
comp3	character 10	actual weapon type or component name in component 3 (if any)
comp4	character 10	actual weapon type or component name in component 4 (if any)
comp5	character 10	(not currently used)
comp6	character 10	(not currently used)
val	float 4	data value

extended to 6. (The files would then have to be more than 80 columns wide, however, if 10-character component names were still used and legibility were to be preserved.) Columns comp5 and comp6 of tables RDATROL and RDATEFF have been set up to accept fifth and sixth components, and the computer code of the aggregator can process variables of up to 6 dimensions. The number of notional dimensions of each variable must not exceed 3--some amount of reprogramming would be necessary to extend this limit.

E. ADAPTIVE INDICES

The aggregator has a certain category of indices, identified by a code number of 10 in INGRES table INDXBINFO, which are called adaptive indices. The aggregator treats these indices somewhat differently from the other types of indices. This section explains the concept of adaptive index and presents basic relevant information. The effects of adaptive indices on the aggregation algorithms are explained in Chapter III, Section H.3.b.4).

1. Rationale

The following example is used to illustrate the adaptive index concept. In NAVMOD, the input variable ENACDT is used in the modeling of Red air and submarine attacks on the (Blue) aircraft carriers. This input is a one-dimensional array, with the definition:

> ENACDT(K)--In Subroutine CTFMOD, expected number of (CVW) aircraft destroyed when an ASM of type K hits a full carrier. K=1,...,NKRB for bombers. K=NKRB+ 1 is for cruise missiles launched from Red submarines. Here, a full carrier excludes those aircraft that, on average, would be out flying missions other than DLI missions.

Different elements of this array can correspond to distinctly different categories of Red platforms: Red bombers and Red missile-firing submarines--the same variable name encompasses data that apply to two different resource categories.

This situation can be adequately modeled by the concepts already explained. The variable K in the definition corresponds to the index MZKRB1, which reflects the type of Red platform attacking the Blue carrier. The index MZKRB1 is a compound resource-based (code-6) index, encompassing the two resource categories Red bombers and Red missile-firing submarines. The more disaggregated data will be stated in terms of the actual types of Red bombers and missile-firing submarines to be considered, rather than the

notional types. (Adding an index on type of carrier may be desirable, as NAVMOD models only one notional type of carrier.)

Suppose the user also wants the more disaggregated data for the variable ENACDT to have an added dimension on the type of Red munition fired at the carrier (in addition to type of Red platform). The resource categories ROLAF, for antiship munitions carried by Red bombers, and ROSCM, for antiship munitions (missiles) carried by Red missile-firing submarines exist. Each of these categories has its associated list (in INGRES table GENTYP0) of actual weapon (munition) types. The user could define a compound resource-based index that would concatenate these two resource categories. The set of component names associated with this index would be the union of the sets of actual types of Red air-launched antiship munitions and Red submarine-launched antiship missiles considered.

In the discussion contained in the preceding section, all combinations of component names associated with different indices are assumed to appear in the more disaggregated data. But here, it does not make sense to consider more disaggregated data that involve combinations of bombers and submarine-launched missiles or submarines and air-launched missiles. If an added index on type of munition is to be specified for the variable ENACDT, one wants to consider only data values for bomber/air-launched-munition and submarine/SLCM combinations. That is, one would want this added index not only to access the actual types of munitions in the resource categories ROLAF and ROSCM but also to adapt the category of munition to the type of platform under consideration.

Subject to certain restrictions (discussed in the following paragraphs), the preprocessor allows the user to define and use such adaptive indices.

2. Structure of Adaptive Indices

a. Basic Definitions and Restrictions

The following remarks should make clear the meaning and form of adaptive indices.

1) There can be at most one adaptive index for a variable. (This is true for the current version of the program. Developing algorithms to treat two or more adaptive indices for a given variable may be possible.)

2) The adaptive index must be an added index; no notional index for a variable is treated as an adaptive index.

3) Associated with each adaptive index is some other index, which shall be called the base index for the adaptive index.

4) This base index must be either a compound resource-based index (index code 4 or 6) or a numerical index with a fixed number of used values (index code 14), without an associated limit variable.

5) If an adaptive index is used as an added index for a variable, its associated base index must be one of the *notional* indices for that variable. (The variable might also have other notional and/or added indices.)

6) Each base index is associated with a sequence of steps. The total number of steps appears in column nsteps of table INDXBINFO, in the row for the *base* index. If the base index is resource-based, each step is associated with a resource category (listed in table INDXGENUS). If the base index is numerical, the steps are simply the numerical values 1, 2, 3,..., up to the number of steps.

7) With each step of the base index, there is an associated resource category for the *adaptive* index. That is, the adaptive index can also be thought of having a series of steps, the same number of steps as the base index. For each step, there is an associated resource category. In general, these resource categories are different for different steps (this is the whole rationale for using adaptive indices), but they need not all be different.

To treat the example contained in the preceding section, an adaptive index would be defined. It can be given any name desired; let us call it JTASCM (ASCM being mnemonic for antiship cruise missile). The associated base index is MZKRB1, a code-6 (compound resource-based) index, with two steps. The resource categories associated with the two steps of the base index are RLBMR (Red land-based bombers) and RSCMA (Red missile-firing submarines). The added adaptive index is to reflect type of munition for such platforms, and thus the resource categories for the two steps of the adaptive index would be ROLAF (antiship munitions carried by Red bombers) and ROSCM (antiship missiles carried by Red missile-firing submarines). (The resource categories for the adaptive index should not be confused with the resource categories for the base index.)

b. Component Names and More Disaggregated Data

Each adaptive index is associated with a sequence of resource categories, one for each step. If the base index is resource based, there is then a sequence of pairs of resource categories, one pair for each step. One of the resource categories in each pair applies to the base index, the other to the adaptive index. Considering the actual resource types associated with each such resource category gives rise to a series of pairs of sets of component names (names of actual resource types) associated with the base-index/adaptive-

index combination. If the base index is numerical, there is a similar series of pairs of sets of component names, but the first set in each pair (i.e., the set for the base index) consists of a single (character-encoded) numerical value.

The structure of the more disaggregated data lines for a variable that has an adaptive index should now be evident. Instead of the single cartesian product of sets of component names (as discussed in Section D), there is a series of such cartesian products, one for each step of the adaptive (or, equivalently, the base) index. For indices other than the base or the adaptive index, the set of component names for that index (as discussed in Section B.4) is used in all the cartesian products. For the base and adaptive indices, the K^{th} pair of sets of component names is used for the K^{th} cartesian product. In each cartesian product, the ordering of the sets of component names is in accordance with the ordering of the indices specified for the variable (in table RVARINFO).

Note that an adaptive index (code 10) differs from a compound resource-based index (code 4 or 6), even though both types of indices have a sequence of resource categories associated with them. The compound resource-based index does not have an associated base index, and the full union set of component names is used in forming the cartesian product for the more disaggregated data lines, regardless of the component names for other indices.[27]

c. INGRES Tables Concerning Adaptive Indices

Each adaptive index must be listed in INGRES table INDXBINFO, with a code number (entry in column indxcode) of 10. The rest of the information in table INDXBINFO is irrelevant for the adaptive index itself (some of it is relevant for the base indices of adaptive indices). Instead, two INGRES tables, named SPLITINDG and SPLITINDX, are used to store information about adaptive indices. Tables II-25 and

[27]Recall that data lines with infeasible combinations of component names--those that simply would not occur in reality--can be eliminated from the aggregation process by specifying a negative data value for them or eliminating them from the data file (see Chapter I, Section B.1.b; also Chapter III, Section H.1). Given this option, it is not necessary, strictly speaking, to use adaptive indices. Instead, an adaptive index can be replaced by a compound resource-based index with the same sequence of resource categories. Let the resulting data lines that have combinations of component names that are not from one of the appropriate pairs of base-index/adaptive-index sets be marked as infeasible. Let the remaining data lines be given the same data values as before. Then the computed aggregated values will be the same as in the adaptive index case. The use of an adaptive index, however, eliminates the work of marking the infeasible data lines.

Table II-25. COLUMNS OF INGRES TABLE SPLITINDG AND THEIR MEANINGS

Column Name	Type and Length	Meaning
indxsplit	character 6	name of adaptive index
indxbase	character 6	name of associated base index

Table II-26. COLUMNS OF INGRES TABLE SPLITINDX AND THEIR MEANINGS

Column Name	Type and Length	Meaning
indxbase	character 6	name of base index
indxsplit	character 6	name of adaptive index
bstep	integer 2	step of base index (1,2,...). Number of steps should equal value in column nsteps of table INDXBINFO for the base index.
genusstep	character 6	name of adaptive index's resource category associated with above step
gvalrl	integer 4	(not currently used)
cumgvalrl	integer 4	(not currently used)

II-26 show the column structure of these tables and the information stored in them. (In future versions of the program, this table and column structure might change slightly.)

Of course, an adaptive index used as an added index for a variable must be specified in INGRES table RVARINFO, as discussed in Section C.1.b. Each resource category associated with a step of an adaptive index must be one of the resource categories set up in the database. The actual resource types to be used as component names are found in tables GENTYP0 and GENUSTYWGT.

At various stages of development of the preprocessor, adaptive indices have been referred to--for evident reasons--as split indices or keyed indices, and some of the computer program variables and INGRES table and column names reflect this. (The term "keyed index," used in this special context, is not to be confused with keys for INGRES tables that have a non-heap storage structure.) Also, the term "keyee" has been used to refer to the base index for an adaptive index.

3. Examples of Possible Adaptive Indices

Subject to the restrictions discussed in the preceding paragraphs, the user can create adaptive indices as desired and can specify that an adaptive index be used as an added index for any effectiveness variable. (None of the indices in the pre-specified lists of added indices for the resource and ordnance stock variables are adaptive indices.)

Several adaptive indices have been set up in the database. Table II-27 defines these indices and indicates the variables for which they have been used, in the current version of the database.[28] Included is the index JTASCM, defined for the example previously discussed. If desired, the user can decide not to employ these indices (the more disaggregated data lines will then have to be changed appropriately; see Chapter IV, Section D.2.c). Tables II-28 and II-29 show the contents of INGRES tables SPLITINDG and SPLITINDX, respectively, that are consistent with the adaptive indices currently defined.

[28] For more complete definitions of these variables, see Appendix F of Reference [2].

Table II-27. POSSIBLE ADAPTIVE INDICES AND VARIABLES THAT MIGHT USE THEM

JDOFST—Blue torpedo, surface-launched or submarine-launched, depending on the step of index MZKBS, which encompasses Blue surface ships and direct-support submarines.

Could be used as an added index for variable:
CPRSCK(KBS)—the probability that a given Blue penetrating ship of type KBS kills a given Red barrier sub, given detection and attack.
Indices accounted for in notional variable:
MZKBS—Blue ship—surface ship plus direct-support submarine
Examples of additional indices:
JRSBAR—Red barrier submarine
JDOFST, as indicated above

JSLBF—Red land-based bombers or fighters. Adapted to the value of the numerical index MYMRLA, used in the variable PKAT1. Value 1 of this index encompasses all types of bombers.

Could be used as an added index for variable:
PKAT1(I)—probability of kill of an attacking bomber (I=1) or escort (I=2) by a salvo of air-to-air missiles fired from task force aircraft.
Indices accounted for in notional variable:
MYMRLA—Numerical index—1 or 2
Examples of additional indices:
JBAFTR—Blue sea-based fighter aircraft
JBOAAA—Blue air-to-air munitions for sea-based aircraft
JSLBF, as indicated above

JTASCM—Red air-launched or submarine-launched antiship munition (missile), depending on the step of the index MZKRB1, which encompasses Red bombers and missile-firing submarines.

Could be used as an added index for variables:
ENACDT(K)—expected number of (CVW) aircraft destroyed when an ASM of type K hits a full carrier. K=1,...,NKRB for bombers, K=NKRB+1 is for cruise missiles launched from Red submarines. Here, a full carrier excludes those aircraft that, on average, would be out flying missions other than DLI missions.
Indices accounted for in notional variable:
MZKRB1—Red antiship missile platform—bomber or submarine
Examples of additional indices:
JTASCM, as indicated above
JBFCAR—Blue carrier

PKPLDT(K)—fraction of incoming ASMs launched from type-K Red bombers that are either defeated by a carrier's passive defense systems or are unreliable for K=1,...,NKRB — K=NKRB+1 is for missiles launched from Red submarines.
Indices accounted for in notional variable:
MZKRB1—Red antiship missile platform—bomber or submarine
Examples of additional indices:
JTASCM, as indicated above

TAB13T(I,K)—factor for reduced carrier effectiveness if a carrier receives I hits by antiship missiles launched from type-K Red bombers, for K=1,...,NKRB. TAB13T(I,NKRB+1) is for missiles from Red submarines.
Indices accounted for in notional variable:
MXTABH—Number of Red missile hits
MZKRB1—Red antiship missile platform—bomber or submarine
Examples of additional indices:
JTASCM, as indicated above

(continued)

Table II-27. (concluded)

JTOFST—Red torpedo, surface-launched or submarine-launched, depending on the step of index MZKRS, which encompasses Red non-barrier ships, including: torpedo-firing submarines, missile-firing submarines (using torpedoes for self-defense), non-resupply surface ships, and resupply ships.

Could be used as an added index for variables:

CPBSCK(KRS)—the probability that a given Red penetrating ship of type KRS kills a given Blue barrier sub, given detection and attack.

Indices accounted for in notional variable:

MZKRS—Red ships

Examples of additional indices:

JBSBAR—Blue barrier submarines

JTOFST, as indicated above

UAPKRS(KRS)—probability that a Red ship of type KRS kills a Blue URG ship that is on its way to or from the task force given that the two ships are in the same region and the Red ship has detected the Blue ship.

Indices accounted for in notional variable:

MZKRS—Red ships

Examples of additional indices:

JBFURG—Blue URG ships

JTOFST, as indicated above

Table II-28. CURRENT CONTENTS OF INGRES TABLE SPLITINDG

indxplit	indxbase
JDOFST	MZKBS
JSLBF	MYMRLA
JTASCM	MZKRB1
JTOFST	MZKRS

Table II-29. CURRENT CONTENTS OF INGRES TABLE SPLITINDX

indxbase	indxplit	bstep	genusstep	gvalrl	cumgvalrl
MZKRB1	JTASCM	1	ROLAF	0	0
MZKRB1	JTASCM	2	ROSCM	0	0
MZKRS	JTOFST	1	ROSTP	0	0
MZKRS	JTOFST	2	ROSTP	0	0
MZKRS	JTOFST	3	ROFTP	0	0
MZKRS	JTOFST	4	ROFTP	0	0
MZKBS	JDOFST	1	BOFTP	0	0
MZKBS	JDOFST	2	BOFTP	0	0
MZKBS	JDOFST	3	BOFTP	0	0
MZKBS	JDOFST	4	BOSTP	0	0
MYMRLA	JSLBF	1	RLBMR	0	0
MYMRLA	JSLBF	2	RLFTR	0	0

III. DESCRIPTION OF SUBROUTINES

Having discussed the structures on which the aggregator methodology is based, we can now describe the aggregator preprocessor's algorithms and operation, in terms of the subroutines of the preprocessor computer program. This chapter details the major subroutines of the preprocessor and provides some discussion of the other subroutines (and the one function subprogram). One emphasis is the information flow between INGRES tables and program variables. Table A-1 of Appendix A contains brief descriptions of all of the program segments of the preprocessor (including the INGRES tables they access), and Table A-3 of that appendix indicates the subroutines that access each given INGRES table.

Recall that the preprocessor has two parts. The first processes the resources, ordnance, and limit variables. It computes aggregated values for these variables. It also computes and stores (in certain places) in the INGRES database weights that will be used in the weighted averaging procedure used for aggregating the effectiveness variables. The second part of the preprocessor determines (from input) which effectiveness variables are to be aggregated and for each such variable performs the averaging of the more disaggregated data into aggregated values. The two parts can be run separately, but if the second part alone is run, the first part must have run previously, so that the weights for averaging have been computed and appropriately stored in the INGRES database.

The first part of the aggregator is performed by Subroutines STINDD, LMVSET, LMVCMP, RSOWGT, and INDWGT. The second part is performed by Subroutines EFFVAR and AGGEFF. All of these subroutines are described in the following paragraphs; first the main program is briefly described.

A. THE MAIN PROGRAM

The main program, AGGPREPRO, starts by initializing the large array ZVALU, which will hold the aggregated values, to all zeros. (The program name AGGPREPRO is longer than the six character ANSI standard, but it can easily be changed if desired.) It then calls several minor subroutines, POWSET, CLSET, and GETIVA, which set up the mechanism that determines the storage locations for the aggregated values. It then calls

Subroutine CHRINI, which encodes the integers 1 through 99 into character strings and stores these encoded values in COMMON block CHRCMN for use by other routines.

The program then connects the database NAVPRE and reads the first line of the input options file, which contains an option code number. If this number is nonzero,[1] the program bypasses its first part and calls Subroutine EFFVAR with the option code specified, to perform the second part, namely, the aggregation of certain effectiveness variables. Otherwise, the program performs its first part--processing the resource, ordnance, and limit variables--as follows.

The program first reads from the input options file the file name and extension of a file containing more disaggregated data values for the resource variables, ordnance stocks, limit variables, and allocation fractions. The device and directory specification for this file is retrieved from INGRES table PFNAMEF; to avoid errors, the user should make sure that the desired data values file does reside in this directory. From these pieces, the program constructs a full file specification. The program then clears out whatever was in INGRES table RDATROL, reads (via the INGRES SQL copy table command) the data values file into table RDATROL, and modifies RDATROL to an ISAM storage structure to speed up retrieval (see Chapter IV, Section A, below, for some implications of this procedure).

The program then opens a file, named RESORDLM.AGD, on which aggregated values for the resource variables, ordnance stock variables, and limit variables will be output. (The output format is such that file RESORDLM.AGD is readable by NAVMOD itself.) The program calls (in order, one time each) Subroutines STINDD, LMVSET, LMVCMP, RSOWGT, and INDWGT, which compute aggregated values of the resource, ordnance stock, and limit variables and also compute and store weights to be used in the aggregation of the effectiveness variables. (These subroutines are all described in the following section.) The file RESORDLM.AGD is closed.[2]

The program then reads another line from the input file. If the end of file has been reached or the option code is 0 again, the program terminates. Otherwise, Subroutine

---

[1] Strictly speaking, neither 0 nor 10; see Chapter IV, Section A.

[2] Strictly speaking, the file RESORDLM.AGD is opened just after the Subroutine STINDD is performed; it is written on by Subroutines LMVSET and RSOWGT; and it is closed just before Subroutine INDWGT is performed.

EFFVAR is called, with the option code as just read, to perform the second part of the program. The program terminates after the second part of the program is performed.

B. SUBROUTINE STINDD ("set indices--detail")

Subroutine STINDD computes normalized weights for those indices called detail indices. These indices, which are identified by a code number of 30 in INGRES table INDXBINFO (see Table II-6) are discussed in Chapter II, Section B.2; a review of that section may be helpful to the user. For each detail index, the user must have filled INGRES table INDXDTYP with the name of the index, the different actual resource types or combat-related names associated with it (in a specified order), and raw (relative) weights for these different types. (The column structure of table INDXDTYP is explained in Chapter II, Section B.3.a.)

Several detail indices have been developed in the current version of the database. Table III-1 contains definitions of these indices, and Table A-8 of Appendix A lists the variables that currently use these indices. Note that a variable can have two or more detail indices as added indices, although this is not the case in the current version of the database. Table III-2 shows how the corresponding INDXDTYP table might appear after the aggregator (Subroutine STINDD) has been run; before the aggregator is run, column rltypwgt values have not been computed. (The numerical values in column rawwgt are intended to be completely hypothetical.) The user can construct whatever other detail indices are desired.

Subroutine STINDD merely computes normalized weights from these raw weights. The normalized weights are stored in column rltypwgt of table INDXDTYP and will be used in Subroutine AGGEFF for variables that have an added detail index. For each detail index, Subroutine STINDD also sets column ivalr1 of table INDXBINFO to the number of combat-related names associated with the index.

C. SUBROUTINE LMVSET ("limit variables set")

The limit variables indicate the numbers of elements of NAVMOD (notional, aggregated) input arrays that will actually be used by NAVMOD, including numbers of types of notional resources for which NAVMOD can model an input number of types. The limit variables play a key role in NAVMOD and in both preprocessors. Subroutine LMVSET retrieves certain information from INGRES table RDATROL and computes

Table III-1. DETAIL INDICES CURRENTLY IN DATABASE

Index Name	Description
JABCML	Blue method of launching land-attack cruise missiles
JABDSM	Blue submarine screen protocol
JABHEL	Blue sea-based ASW helicopter type
JARHEL	Red sea-based ASW helicopter type
JARSBA	Red sea-based (fighter) aircraft

Table III-2. ILLUSTRATIVE CONTENTS OF INGRES TABLE INDXDTYP

indxnam	ordno	rltyp	rawwgt	rltypwgt
JABCML	1	VERTLAUNCH	0.300	0.300
JABCML	2	ARMOREDBOX	0.300	0.300
JABCML	3	TORP. TUBE	0.400	0.400
JABDSM	1	FIXED BAR.	0.200	0.200
JABDSM	2	MOVING BAR	0.600	0.600
JABDSM	3	AREA CLEAR	0.200	0.200
JARHEL	1	HORMONE	10.000	0.500
JARHEL	2	HELIX	10.000	0.500
JARSBA	1	Ntl.RdSBAC	1.000	1.000
JABHEL	1	SH-2	4.000	0.500
JABHEL	2	SH-3	2.000	0.250
JABHEL	3	SH-60	1.000	0.125
JABHEL	4	CV HELO	1.000	0.125

values for the limit variables. The information about limit variables is stored in INGRES table LIMVINFO. Table III-3 defines the columns of table LIMVINFO, and Table III-4 shows its contents before the aggregator has been run. These contents should not be changed by the user unless NAVMOD is changed and then only with great care. Subroutine LMVSET updates column lvval of table LIMVINFO. The rest of the information has been appropriately preset. For definitions of the limit variables, see Table II-19.

The subroutine retrieves and processes each row of the table, in turn. It first reads all of the entities in the row. For the one case (Red cruise-missile-firing submarine) that has no limit variable name, lvval is set to dimval (which equals the fixed number of subtypes--four--that NAVMOD can model). Otherwise, the subroutine searches INGRES table RDATROL for a value for the limit variable with name lvnam; it searches table RDATROL for a record with the current limit variable name in the vnamer field and when it finds it, retrieves the value in the val field into the working variable LVAL1. (See Chapter II, Section D for information on table RDATROL.) If no such record is found, LVAL1 is set to 1.

Some limit variables are associated with a resource category (column genus of table LIMVINFO) and an allocation fraction variable (column allocvar). Recall (Chapter II, Section C.3.c) that the allocation fraction variable f_{ij} specifies the fraction of actual resource type i that is to be considered as notional resource type j (e.g., the fraction of CG-47s that will be considered to be Blue notional-type-1 escort ships). The quantity

$$\max \{j \mid f_{ij} > 0\}$$

corresponds to the largest notional type of resource that will be considered. Accordingly, if the limit variable currently under consideration has an associated resource category and allocation fraction variable,[3] Subroutine LMVSET searches table RDATROL for the indicated maximum and stores the result in the working variable LVAL2. For maximum consistency of results, LVAL1 and LVAL2 should have the same value. If they do not, a

[3] Except for category RSCM--Red cruise missile-firing submarines--which has no limit variable and is treated as discussed previously.

Table III-3. COLUMNS OF INGRES TABLE LIMVINFO AND THEIR MEANINGS

Column Name	Type and Length	Meaning
lvnam	character 6	name of limit variable (can be blank for code 3 and code 4 resource categories, which should be listed in this table)
lvval	integer 4	value of limit variable (if lvnam is blank, then number of types of resource simulated; should agree with table GENUSINFO)
dimnam	character 6	name of symbolic constant associated with limit variable or resource category
dimval	integer 4	value of such symbolic constant
genus	character 6	code name of resource category, if any, associated with limit variable
allocvar	character 6	name of "allocation variable" governing assignment of real types in the resource category to notional types. Blank if no associated resource category.

Table III-4. CONTENTS OF INGRES TABLE LIMVINFO

lvnam	lvval	dimnam	dimval	genus	allocvar
NABSAM	0	MXABSM	10	RMABD	FABRSM
NKBDPL	0	MXKBD	11	BLFTR	FPLBLB
NKBES	0	MXKBES	10	BFESC	FBESS
NKRB	0	MXKRB	10	RLBMR	FATABT
NKRSN	0	MXKRSN	11	RFNRP	FRSFNR
NLOC	0	MXLOC	12		
NPPSAM	0	MXPPSM	10	RMPPD	FPPRSM
NSAG	0	MXSAG	13		
	4	MYKRSG	4	RSCM	FRSSG
NMPSG	0	MXMPSG	10		
MIMP	0	MXNMP	90		
NTABA	0	MXTABA	20		
NTABD	0	MXTABD	12		
NTABH	0	MXTABH	20		
NPPFFX	0	MXTABP	20		
NTABS	0	MXTABS	20		
NTABV	0	MXTABV	20		
NTC	0	MXTC	7		

message is printed on the output file of informative messages (associated with logical unit 4).

If the limit variable under consideration does not have an associated resource category, LVAL2 is set to 0. The code then computes a final value

$$LVVAL = \max\{LVAL1, LVAL2, 1\}$$

for the limit variable and sets column lvval of table LIMVINFO to the value LVVAL, for the row currently under consideration. A number of entries of other INGRES tables are also updated to this value. Except for the limit variables NKRSN and NMPSG, a data record with the limit variable name and computed value is written on the output file for aggregated variables, RESORDLM.AGD. (Subroutines FINDVN, PUTONE, and GETALL are used to generate this data record). The NAVMOD input NKRS is computed as NKRSN+3, and each element of the NAVMOD input array MIMPSG is set to the value of NMPSG.[4] Data records for NKRS and MIMPSG are then generated and written on file RESORDLM.AGD.

D. SUBROUTINE LMVCMP ("limit variables--computed")

As indicated in Chapter II, Section B.2, some of the indices used by the aggregator encompass a sequence of several different resource categories, and some numerical indices consist of a sequence of other numerical indices. The code numbers in column indxcode of table INDXBINFO for these compound indices are 4, 6 and 16 (see Table II-6). For each compound index with code 6 or 16, Subroutine LMVCMP computes a number of notional types (types that NAVMOD will model) and stores it in column ivalntl of table INDXBINFO.[5] These numbers are later used in Subroutine AGGEFF, to help specify the number of notional values (NAVMOD input data elements) for which an aggregation procedure is to be performed.

For each code-6 index, the set of component resource categories is stored in table INDXGENUS. Each resource category has an associated number of notional types (i.e., types that NAVMOD can model), which was preset to the appropriate fixed number[6] or

---

[4]The aggregator is not currently capable of handling limit variables, such as MIMPSG, that are arrays. MIMPSG can be reset via the interaction selector preprocessor utilities or via editing the appropriate data file. (See Chapter II, Section C.3.b.)

[5] The indices with code 4 are characterized by a fixed number of notional types; no computation is needed.

[6] Generally, 1, but 4 in the case of the number of subtypes of Red cruise-missile-firing submarines.

has been set to the limit variable value just computed in Subroutine LMVSET. Subroutine LMVCMP merely retrieves the sum of these numbers of notional types (over the appropriate set of component resource categories) and uses this sum as the number of notional types for the code-6 index.

The component numerical indices for code-16 indices are stored in INGRES table CMPLIM. Along with each such component index is stored a number of notional values to be used; this is either preset or has been set (by Subroutine LMVSET) to the value of a limit variable just computed. Subroutine LMVCMP retrieves the sum of these numbers of notional values for the component indices and uses this sum as the notional value of the code-16 index.

Currently, there are only two code-16 indices, MZLOC1, one more than the number of regions played, and MZIPR, a numerical index corresponding to notional types of Red land-based aircraft. MZLOC1 is an index for variable PRSM, and MZIPR is an index for variable IPRSRA (see Appendix F of Reference [2] for definitions of variables). The computer code, however, provides a general methodology; the specific information is only in the database.

E. SUBROUTINE RSOWGT ("resources, ordnance, and weights")

1. Introduction

Subroutine RSOWGT reads more disaggregated data values from the INGRES table RDATROL and from these data produces two main groups of information.[7,8] The first group consists of data values for the NAVMOD (notional) resource and ordnance stock variables; these values are output on the file RESORDLM.AGD, in such a format that the file can later be read by NAVMOD. The second group comprises sets of weights, one weight for each combination of actual resource type and notional (NAVMOD) resource type. These weights are stored in INGRES table GENUSTYWGT and are used later in the

[7] The information on more disaggregated data values for the resource variables, ordnance stock variables, limit variables, and allocation fractions has been read into INGRES table RDATROL from a file, in the main program (under input option code 0) or has been established in table RDATROL prior to running the program (under input option code 10). All of the more disaggregated data values for resources, ordnance stocks, limit variables, and allocation fractions are assumed to be nonnegative.

[8] For definitions of the columns of the three tables RDATROL, GENUSTYWGT, and RESGENUS, which Subroutine RSOWGT uses frequently, see Tables II-24, II-4, and II-5, respectively.

program for the weighted averaging of effectiveness variables (Subroutine AGGEFF).[9] This part of Subroutine RSOWGT computes the weights for aggregation of effectiveness variables from the resource variables. This approach is used because the effectiveness parameters apply to the resources and vary as a function of resource type. Thus the relative weight to be given to a certain resource type might reasonably depend on the number of resources of that type present.

An attempt has been made to keep the computer code of Subroutine RSOWGT as general as possible; the existence of some broad patterns in the NAVMOD resource variable structure has allowed this. (See Table II-22 of Chapter II). Nonetheless, the code uses some individual resource and limit variable names; if these variables change in NAVMOD, the RSOWGT code might also have to be changed accordingly. (Certain database tables would also require changes.) The code also assumes certain specified indexing patterns for the resource variables. If the user wishes to change the indexing of the resource variables (the NAVMOD indexing and/or the added indices), then certain code and/or database changes might be necessary. The details of Subroutine RSOWGT are explained in the following section.

2. Computational Procedure

After initializing certain variables and INGRES table columns, Subroutine RSOWGT performs a series of computations for each row of INGRES table RESGENUS, in turn. Table RESGENUS is a heap file, and its rows can be processed in any order.

The following description applies to a single row. First, the information in the row is retrieved into program (FORTRAN) variables. (The variable names are the same as the names of the corresponding columns of table RESGENUS, except column genusnam goes to variable GENNAM.) These variables are

IMETH:	the number of the computation method to be used
RESNAM:	the name of the resource (or ordnance stock) variable
GENNAM:	(column genusnam) name of the associated resource category

[9] The contents of INGRES table GENUSTYWGT have been constructed so that every feasible resource category-actual type-notional type combination is represented in some row. Subroutine RSOWGT fills in the weights. If the sets of actual resource types (within the resource categories) change or if the maximum numbers of notional types change, the auxiliary program GENGENUST can (and should) be run to regenerate table GENUSTYWGT so that its contents are consistent with the new structure of types.

ADLVAR: name of an additional variable to be used in the processing, if any

IMUNI: a numerical value that is sometimes used in the processing.

Table III-5 shows the contents of table RESGENUS. For definitions of the variables listed, review the tables in Chapter II, Section C. The resource category names are defined in Table II-1.

For computation methods 4, 5, and 6, the additional variable ADLVAR is an allocation fraction; for methods 7 and 8 it is a reserve resource stock. The use of ADLVAR and IMUNI depends on the specific computation method (which is discussed in Section 3). For rows where IMETH=9, a resource category is not applicable; see Section 3.i.

After retrieving the row information from table RESGENUS, Subroutine RSOWGT calls Subroutine CLRWK to initialize certain working variables to zero. Then, for the resource category under consideration (in variable GENNAM), Subroutine RSOWGT retrieves, in order, all of the actual resource types included in this category. (The lists of types are stored in INGRES tables GENTYP0 and GENUSTYWGT.) The names of the types are assigned, in order, to elements of the character array RLTYP, and the number of types is assigned to the variable NRLTYP. If the number of types retrieved is zero, a message is printed on the file associated with logical unit 4, and processing continues. This message indicates a potential error, except in the case IMETH=9, where no resource category is associated with the resource variable.

Subroutine FINDVN is called, to obtain the order number of the resource variable in the list of NAVMOD inputs. As mentioned earlier, the variables RSBTP, RSFNRP, and RSFRSP are not in themselves NAVMOD inputs; in this case, FINDVN returns a value of zero, but the computer code has explicit special treatments for these variables.

The routine then computes the relevant data values for NAVMOD and weights for aggregation of effectiveness variables, using the computation method indicated by variable IMETH. (The individual computation methods are described in the following section.) Each method is tailored to a different indexing structure for resource variables; these structures have been presented in Table II-22. Each method (except Method 9) uses the list of actual resource types in array RLTYP. As notional (NAVMOD) data values for the resource variable are computed, they are stored in a large array (named

Table III-5. CONTENTS OF INGRES TABLE RESGENUS

imeth	resnam	genusnam	adlvar	imuni
1	BSSNDS	BSDS		0
1	OIBUA	BUA		0
1	OIBUR	BUR		0
1	OVBLAA	BOLAA		0
1	OVBLAF	BOLAF		0
1	OVBLAS	BOLAS		0
1	OVBUL	BUL		0
1	OVRLAA	ROLAA		0
1	OVRLAF	ROLAF		0
1	XAEW	BAAEW		0
1	XASW	BAASW		0
1	XATTCK	BAATT		0
1	XFGHTR	BAFTR		0
1	XPLAT	BFCAR		0
2	BSIBAR	BSBAR		0
2	RSBTP	RSTP		0
2	RSFRSP	RFRSP		0
2	RSIBAR	RSBAR		0
2	TOBBAR	BFBT		0
2	TORBAR	RFBT		0
2	XAEWLQ	BLAEW		0
2	XASWLQ	BLASW		0
3	AESCAB	RLFTR		0
3	AINTCP	RLINT		0
4	ABRSAM	RMABD	FABRSM	0
4	BESS	BFESC	FBESS	0
4	PPRSAM	RMPPD	FPPRSM	0
4	ABANM	RMABD	FABRSM	1
4	PPANMS	RMPPD	FPPRSM	1
5	PLBLBD	BLFTR	FPLBLB	0
5	RSFNRP	RFNRP	FRSFNR	0
5	RSSG	RSCM	FRSSG	0
6	ATABT	RLBMR	FATABT	0
7	XURGS	BFURG	RURGS	0
7	OIBSTP	BOSTP	OVBSTP	1
7	OIBSCM	BOSCM	OVBSCM	2
7	OIBFCM	BOFCM	OVBFCM	3
7	OIBFTP	BOFTP	OVBFTP	4
7	OIBFFF	BOFFF	OVBFFF	5
7	OIBFDA	BOFDA	OVBFDA	6
7	OIBFDS	BOFDS	OVBFDS	7
7	OIBAAS	BOAAS	OVBAAS	8
7	OIBAAA	BOAAA	OVBAAA	9
7	OIBAAF	BOAAF	OVBAAF	10
7	OIBAAG	BOAAG	OVBAAG	11
8	OIBBAR	BOBAR	OVBBAR	0
8	OIRBAR	ROBAR	OVRBAR	0
8	OIRSTP	ROSTP	OVRSTP	1
8	OIRSCM	ROSCM	OVRSCM	2
8	OIRFFL	ROFFL	OVRFFL	3
8	OIRFFR	ROFFR	OVRFFR	4
8	OIRFDL	ROFDL	OVRFDL	5
8	OIRFDA	ROFDA	OVRFDA	6
8	OIRFDS	ROFDS	OVRFDS	7
8	OIRFTP	ROFTP	OVRFTP	8
8	OIRFAS	ROFAS	OVRFAS	9
8	OIRFAA	ROFAA	OVRFAA	10
9	RUNRCP			0
9	SHEL			0

ZVALU), at a position computed by Subroutine PUTONE. After all of the notional values have been computed, Subroutine GETALL is called (for that variable) to output the values on the file RESORDLM.AGD (unit 17), in a format such that the output file can later be read by NAVMOD. The aggregation weights are stored in table GENUSTYWGT (as described in the next section).

After all rows of table RESGENUS have been processed, Subroutine RSOWGT calls Subroutine GETALL to output (on file RESORDLM.AGD) the notional values for NAVMOD resource variables RS, OVBTF, and OVRRG (each of which combines several resource categories and encompasses several rows of table RESGENUS; see Chapter II, Section C.3.d). Control then returns to the main program, which closes the RESORDLM.AGD file.

3. Computation Methods for Resources and Weights

This section provides algebraic descriptions of the weight and data computations, which form the heart of Subroutine RSOWGT. Different computation methods correspond to different indexing structures for the resource variables under consideration, as presented in Table II-22 of Chapter II.

The following notation is used throughout this section:

- i indexes actual type of resource; it ranges from 1 through the number of actual resource types in the resource category being considered; different values of i correspond to different resources in the category, in the order indicated in table GENTYP0;
- j indexes notional resource type, i.e., the type (or subtype) used in NAVMOD (for resources that have more than one such type);
- ι indexes region (the symbol ι is to be read as a lower case "ell"); it ranges from 1 through the number of regions played (limit variable NLOC, which Subroutine LMVSET has determined);
- k indexes Red airbase type; k=1 corresponds to vulnerable airbases, k=2 to invulnerable airbases;
- I is the upper limit for i, i.e., the number of actual resource types in the category being considered; this value has been determined and is stored in working variable NRLTYP;
- J is the upper limit on j, i.e., the total number of notional types. If $J \neq 1$, its value is retrieved from table LIMVINFO, for the resource category under consideration. Except for the case of Red missile-firing submarines, for which NAVMOD simulates a fixed number of subtypes, J is the value of some limit variable;

L is the upper limit on ι and equals the value of the limit variable NLOC.

The following correspondences between the above algebraic notation and the FORTRAN notation of the computer code occur frequently:

i -- IRTY or ITY,

j -- INTY,

ι -- LOC,

k -- IAB,

I -- NRLTYP,

J -- NNTLTY,

L -- NLOC.

Each of the following subsections introduce additional notation, which should be regarded as unique to its subsection, although certain patterns will be evident. The working variables WKVEC, WKARY1, and WKARY2 are used to store many of the quantities mentioned below, until the database is updated and/or the notional variables are output. The reader may wish to refer back to Table III-5 for information on the specific variable and category names used, and to Tables II-1 and II-22, in Chapter II, for definitions of these variables.

Before discussing the specific algebraic algorithms of Subroutine RSOWGT, a few words about storage of the aggregation weights are in order. These weights are stored in INGRES table GENUSTYWGT, the column structure of which is explained in Table II-5. In the following subsections, the computation of raw or relative weights v_i and v_{ij} and normalized weights w_i and w_{ij} are explained. Given that table GENUSTYWGT has been set up correctly before the aggregator is run, there should be exactly one row of table GENUSTYWGT where:

- the value in column genusnam equals the name of the resource category under consideration
- the value in column ordno equals the value of i
- the value in column ntlindval equals the value of j, if there is a j index on the working variable. If there is not a j index, then the value in column ntlindval should be 1.

Then in this row, the value in column rawwgt is set to v_i or v_{ij} and the value in column rltypwgt is set to w_i or w_{ij}.

a. Computation Method 1

In this method, the NAVMOD resource is a scalar; the more disaggregated resource is indexed by actual type. Examples are Blue carriers and sea-based aircraft, and ordnance stocks for land-based aircraft. For each actual type i, the number of units, a_i, of that type is retrieved from table RDATROL. The notional resource value is computed as

$$x = \sum_{i=1}^{I} a_i$$

The raw weight v_i is simply equal to a_i for each i, and the normalized weight w_i is defined as

$$w_i = \begin{cases} v_i/x & \text{if } x > 0\,; \\ 1/I & \text{if } x = 0\,. \end{cases}$$

b. Computation Method 2

In this method, the notional resource varies by region. The actual resource has this region index, plus an index on actual resource type. Examples are Blue and Red barrier submarines and tenders. The resource categories Red torpedo-firing submarines in regions and Red resupply ships also use this method. As indicated in Chapter II, Section C.3.d, above, special resource variable names are used to indicate the more disaggregated data values for these resources. The computed notional values are assigned to appropriate components of the NAVMOD input array RS.

From table RDATROL, the program retrieves $a_{\iota i}$, the number of units of actual resource type i associated with region ι, for the resource category being considered. The NAVMOD resource variable for region ι is computed as

$$x_\iota = \sum_{i=1}^{I} a_{\iota i} \qquad \iota=1,\ldots,L\ .$$

The raw weight for actual resource type i is defined as

$$v_i = \sum_{\iota=1}^{L} a_{\iota i} \qquad i=1,\ldots,I\ .$$

Let $\bar{v} = \sum_i v_i$. The normalized weight w_i is defined, for each i, as

$$w_i = \begin{cases} v_i/\bar{v} & \bar{v} > 0\,; \\ 1/I & \bar{v} = 0\,. \end{cases}$$

c. Computation Method 3

This method is identical to Method 2, except an index k on airbase type (1 = vulnerable airbases; 2 = invulnerable airbases) is used instead of an index (ι) on region. This method is used for Red (land-based) fighter and interceptor aircraft.

d. Computation Method 4

This method is used for resources that have a variable number of notional types (the number of such types being indicated by a limit variable). The NAVMOD resource variable has one index, on notional resource type. The more disaggregated data use actual resource types instead of notional resource types. An allocation fraction variable, the name of which is given in column adlvar of table RESGENUS (for method-4 resources), specifies the apportionment of actual resources to notional resource types--f_{ij} is the fraction of the actual-type-i resources that are to be considered notional type j. (The reader might wish to review Chapter II, Section C.3.c.) Examples of method-4 resources are Blue escort ships and Red ground-based SAMs. The variables ABANM and PPANMS, which represent ordnance for Red ground-based SAMs, are considered to be method-4 resources, but the entry in column imuni of table RESGENUS is 1, not 0, indicating special treatment. For now, assume that one of the other method-4 resources is being processed.

The algorithm begins by retrieving J, the number of notional types of the resource, from table LIMVINFO (this value is in column lvval, in the row where the entry in column genus equals the name of the resource category under consideration). The values of a_i, the number of units of actual resource type i, and the values of the allocation fractions f_{ij} are retrieved from table RDATROL. Recall that the allocation fraction values are assumed to satisfy

$$\sum_{j=1}^{J} f_{ij} = 1, \qquad i=1,\ldots,I\ .$$

Define

$$v_{ij} = f_{ij} a_i \qquad i=1,\ldots,I;\ j=1,\ldots,J\ .$$

This quantity represents the number of units of actual resource type i that are to be considered as notional resource type j. For j=1,...,J, the number of units of notional resource type j is computed as

$$x_j = \sum_{i=1}^{I} v_{ij} = \sum_{i=1}^{I} f_{ij} a_i .$$

This value is assigned to array element j of the appropriate NAVMOD input variable.

A set of relative weights and a set of normalized weights are computed for each notional type j. Each set has one weight for each actual type of resource and represents the relative mix of actual resources that compose a notional resource NAVMOD models. The weights are computed as follows. The quantity

$$v_{ij} = f_{ij} a_i$$

is used as the relative weight for actual resource type i within the set for notional resource type j. Recall that we defined $x_j = \sum_{i=1}^{I} v_{ij}$. The normalized weight w_{ij} is defined as

$$w_{ij} = \begin{cases} v_{ij}/x_j & x_j > 0 ; \\ 1/I & x_j = 0 . \end{cases}$$

Additional sets of weights are computed for the total amounts of actual resources, summing over notional type. Since $\sum_{j=1}^{J} f_{ij}$ is assumed to equal 1, then

$$\sum_{j=1}^{J} v_{ij} = \sum_{j=1}^{J} f_{ij} a_i = a_i \qquad i=1,...,I .$$

The code defines the total raw weights

$$v_{i0} = a_i , \qquad i=1,...,I;$$

the sum

$$\bar{v}_0 = \sum_i a_i ;$$

and the total normalized weights

$$w_{i0} = \begin{cases} v_{i0}/\bar{v}_0 & \bar{v}_0 > 0, \\ 1/I & \bar{v}_0 = 0, \end{cases}$$

for $i=1,...,I$.

For each (i,j) pair (including cases where i=0, as defined above), v_{ij} is stored in column rawwgt of table GENUSTYWGT and w_{ij} is stored in column rltypwgt of table GENUSTYWGT. These weights are used later by Subroutine INDWGT (as well as by Subroutine AGGEFF).

The resource variables ABANM and PPANMS represent munitions (missiles) for Red ground-based airbase defense and power-projection-defense SAMs, respectively. In both NAVMOD and the more disaggregated data, it is assumed that each type of SAM uses its own type of munition. In the more disaggregated data, there are no indices explicitly for Red ground-based SAM munitions; rather, the index on type of SAM is used (indices MXABSM and MXPPSM). Subroutine RSOWGT accepts more disaggregated values of ABANM and PPANMS, as functions of actual SAM type, and computes (via Method 4) notional values using the same allocation fraction variables FABRSM and FPPRSM used for the Red air base defense and power-projection-defense SAMs themselves. The weight computations are not performed, however. The value of 1 in column imuni of table RESGENUS for the rows ABANM and PPANMS is an indicator for the code to bypass the weight computation sections.

e. Computation Method 5

This method combines computation Methods 2 and 4. It is used for resource variables for which the first dimension is an index on resource type (notional type for the NAVMOD input, actual resource type for the more disaggregated data), and the second dimension is an index on region. The resources currently treated by this method are Blue land-based fighter aircraft, Red missile-firing submarines, and Red surface ships in regions, excluding resupply ships. The latter two categories have some category-unique treatments, which are explained later in this section.

The computation method starts by reading the value of J, the number of notional types, from table LIMVINFO, as in Method 4. Then, the code reads (from table RDATROL) values for $a_{i\iota}$, the number of units of actual resource type i (in the category under consideration) associated with region ι, for $i=1,...,I$ and $\iota=1,...,L$. From these $a_{i\iota}$, the sums

$$v_{i0} = \sum_{l=1}^{L} a_{il} \qquad\qquad i=1,\ldots,I$$

and

$$\bar{v}_0 = \sum_{i=1}^{I} v_{i0} = \sum_{i=1}^{I} \sum_{l=1}^{L} a_{il}$$

are computed. These total raw weights have the same interpretation as in Method 4. Normalized total weights are computed, for i=1,...,I, as

$$w_{i0} = \begin{cases} v_{i0}/\bar{v}_0 & \bar{v}_0 > 0\,; \\ 1/I & \bar{v}_0 = 0\,. \end{cases}$$

The v_{i0} and w_{i0} are stored in INGRES table GENUSTYWGT, as in Method 4.

NAVMOD models Red missile submarines as forming one notional type, which is divided into four notional subtypes (see Reference [1], Chapter II, Section B.5). Two different resource categories, RSCA and RSCMA, represent the missiles when divided into notional subtypes and the missiles considered as one notional type. The resource category RSCMA is a part of some indices. The same set of actual missile types is used for both RSCM and RSCMA.

The v_{i0} and w_{i0} above are computed for category RSCM. The code then defines, for i=1,...,I

$$v'_i = v_{i0}$$

$$w'_i = w_{i0}$$

and stores them in columns rawwgt and rltypwgt of GENUSTYWGT, in the rows where

- the entry in column genusnam equals 'RSCMA',
- the entry in column ntlindval equals 1, and
- the entry in column ordno equals i, for each value of i in turn.

Currently, these resource categories, RSCM and RSCMA, are referred to by name in the RSOWGT computer code. A general algorithm to treat resource categories with notional subtypes might be developed, but this has not yet been done.

For all method-5 resource categories, the code proceeds to treatment of individual notional types. In Method 5, as in Method 4, an allocation fraction variable, denoted here

by f_{ij} (the actual variable name is stored in table RESGENUS), represents the fraction of the resources of actual type i that are to be considered as notional type j. For each i the condition $\sum_{j=1}^{J} f_{ij} = 1$ should hold. The following procedure is performed for each notional type j in turn (for j=1,...,J).

The code reads f_{ij} from table RDATROL. The allocation fraction is independent of region, and for each region ι, the quantity $a_{i\iota}f_{ij}$ represents the number of actual type-i resources in region ι that are going to be considered as notional type-j resources--again, in region ι. The total number of notional type-j resources in region ι is computed as

$$x_{j\iota} = \sum_{i=1}^{I} a_{i\iota}f_{ij} \qquad \iota=1,\ldots,L.$$

This value is stored and is later generated as output on the file RESORDLM.AGD. For Red non-resupply ships, the resource variable RSFNRP is not itself an input to NAVMOD, so the code assigns the computed notional value to the appropriate element of the NAVMOD input array RS.

In computing the relative weights for the actual resources within the category, the code takes sums over regions. That is, the code computes, for each j = 1,...,J,

$$v_{ij} = \sum_{\iota=1}^{L} a_{i\iota}f_{ij}, \qquad i = 1,\ldots,I,$$

$$\bar{v}_j = \sum_{i=1}^{I} v_{ij}, \text{ and}$$

$$w_{ij} = \begin{cases} v_{ij}/\bar{v}_j & \bar{v}_j > 0 \\ 1/I & \bar{v}_j = 0 . \end{cases}$$

(Note that $\bar{v}_j = \sum_{\iota=1}^{L} x_{j\iota}$.) These weights are stored in table GENUSTYWGT, as in Method 4.

In the computer code, these computations are performed inside a loop on notional type j, and some of the working variables that hold certain quantities discussed in this section are re-used in successive passes of the loop.

f. Computation Method 6

Computation Method 6 is nearly identical to Method 5, except the index ι on region is replaced by an index k on airbase type (1--vulnerable airbases, 2--invulnerable airbases). The index k is the first index on the resource, so one would think of a_{ki} instead of $a_{i\iota}$; the notional resource would be x_{kj}. An allocation fraction variable is used to apportion actual resource types to notional resource types. Mutatis mutandis, all computations proceed as in Method 5.

Currently, only one resource category--Red bombers--is treated by Method 6.

g. Computation Method 7

This method is used for ordnance stocks on Blue surface ships and direct-support submarines and for Blue URG ships. Two variable names are involved, both stored in table RESGENUS. In column resnam (of table RESGENUS) appears the name of an active stock variable and in column adlvar appears the name of a corresponding reserve stock variable. In NAVMOD itself, the active stock variable is a scalar; the more disaggregated data adds an index on actual resource type. The reserve stock variable RURGS is used in NAVMOD for reserve Blue URG ships. In NAVMOD it is a scalar, in the more disaggregated data, it has an index on URG ship type. The other reserve stock variable names have been developed as indicated in Table II-21. From the values for each such variable, a notional reserve ordnance stock value is computed and is stored in an element of the NAVMOD array OVBTF. The particular element is indicated in column imuni of table RESGENUS; see Table III-5. The entry in column imuni for the Blue URG ship row is zero. Although the name OVBTF is used, an attempt has been made to keep the code as generalizable as possible, in accordance with the following algebraic description.

The computations of Method 7 are as follows. First, the values of the active stock a_i and the reserve stock b_i are read from table RDATROL, for each actual resource type i (included in the resource category indicated in table RESGENUS). The notional active and reserve stock values are computed respectively as

$$x = \sum_{i=1}^{I} a_i$$

and

$$y = \sum_{i=1}^{I} b_i .$$

The quantity x is assigned to the NAVMOD notional variable for the active stock; the quantity y is assigned to the appropriate NAVMOD array element for the reserve stock, as indicated above.

To form the weights for the actual resource types within the resource category, both the active and reserve stocks are considered. The relative weight is defined as

$$v_i = a_i + b_i \qquad\qquad i=1,\ldots,I \; .$$

Letting

$$\bar{v} = \sum_{i=1}^{I} v_i ,$$

the code defines the normalized weights

$$w_i = \begin{cases} v_i/\bar{v} & \bar{v} > 0 , \\ 1/I & \bar{v} = 0 . \end{cases}$$

These weights, v_i and w_i, are stored in INGRES table GENUSTYWGT, in columns rawwgt and rltypwgt, respectively, in the rows where

- the entry in column genusnam equals the name of the resource category under consideration,
- the entry in column ntlindval equals 1, and
- the entry in column ordno equals i,

for each i in turn.

h. Computation Method 8

Method 8 is similar to Method 7, in that it treats both active and reserve variables. There is one notional type of the resource under consideration. The notional active variable is a vector indexed by region; the notional reserve stock is a scalar. The more disaggregated data adds an index on an actual type of resource. This method is used for Blue and Red barrier submarine ordnance stocks, and ordnance stocks for Red surface ships and submarines in regions. For these latter ordnance stocks, the variable name in table RESGENUS for the reserve stock is not used in NAVMOD itself (see Table II-21 for

definitions of these variables). The computed notional value is assigned to an element of the NAVMOD input array OVRRG. The particular element is indicated by the entry in column imuni of table RESGENUS. (This treatment is similar to the treatment of Blue ordnance in Method 7.)

The actual computations are as follows. From table RDATROL, the program reads

$a_{\iota i}$ = the number of units of actual resource type i in the active stock in region ι, and

b_i = the number of units of actual resource type i in the reserve stock,

for i=1,...,I, and ι=1,...,L. The notional active stock in region ι is computed as

$$x_\iota = \sum_{i=1}^{I} a_{\iota i} ,$$

and the notional reserve stock is computed as

$$y = \sum_{i=1}^{I} b_i .$$

These values are assigned to the appropriate NAVMOD variables. The relative and normalized weights are computed using a sum over region, as in Method 5, and combining reserve and active stocks, as in Method 7. Namely:

$$v_i = b_i + \sum_{\iota=1}^{L} a_{\iota i} ,$$

$$\bar{v} = \sum_{i=1}^{I} v_i ,$$

$$w_i = \begin{cases} v_i/\bar{v} & \bar{v} > 0 , \\ 1/I & \bar{v} = 0 . \end{cases}$$

The v_i and w_i are stored in table GENUSTYWGT, in the same manner as Method 7.

i. Computation Method 9

This method is used for two NAVMOD resources, aircraft shelters and runway repair material, for which resource categories (with sets of actual resource types) have not been established in the aggregator preprocessor. The more disaggregated data consist of one scalar value for each variable. Subroutine RSOWGT reads the value from table RDATROL and simply copies it to the corresponding NAVMOD variable. Since sets of

actual resource types have not been developed for these variables, no weight computation is performed.

F. SUBROUTINE INDWGT ("indices and weights")

Certain indices used by the aggregator are associated with a sequence of resource categories--these indices have code number 4 or 6 in (column indxcode of) table INDXBINFO. It might be desirable to specify such a compound-resource-based index as an added index for some effectiveness variable(s). The purpose of Subroutine INDWGT is to compute sets of weights for these compound indices, to be used in the weighted averaging routine (Subroutine AGGEFF) when a compound index is used as an added index.

The weights are stored in INGRES table INDXRTYP. As shown in Table II-11, this table has the following columns:

indxnam	=	name of index
ordno	=	order number (1,2,3,...) of actual resource type within this index;
rltyp	=	10-character code name of actual resource type;
rawwgt	=	relative weight for this resource type for this index; and
rltypwgt	=	normalized weight for this resource type for this index.

The information in the first three columns is preset before the aggregator is run.[10] The values in the latter two columns are computed by Subroutine INDWGT.

Each compound index encompasses several resource categories. Each resource category is associated with a set of actual resource types (this set is stored in tables GENTYP0 and GENUSTYWGT). The set of resource types associated with the code-4 or code-6 index (in table INDXRTYP) is the union (eliminating duplicate resource types) over all component resource categories in the index of the sets of resource types associated with each resource category. It is possible that a given (actual) resource type might appear in two or more different resource categories that are included in the same compound index. For example, the index MYKBA encompasses Blue sea-based attack aircraft and Blue sea-based fighter aircraft. The F/A-18 might be a resource type in both categories. When MYKBA is used as an added index, the relative weight for F/A-18s in that index takes into

---

[10]The database can be delivered with this information correctly set up. If it is desired to change the actual resource types considered, the auxiliary program GENIXRTYP should be run to reset the information in table INDXRTYP appropriately.

account both F/A-18s that are considered notional attack aircraft and F/A-18s that are considered fighter aircraft. (If this does not adequately reflect the reality of the situation, the user might wish to use the single-resource-category indices JBAATT (Blue sea-based attack aircraft) or JBAFTR (Blue sea-based fighter aircraft) as an added index instead of MYKBA. The user can choose added indices for effectiveness variables as desired.)

The cardinality of the above union set (i.e., the associated number of distinct actual resource types) is stored in column ivalrl of table INDXBINFO, in the row for the particular index being considered.

Subroutine INDWGT starts by initializing all elements of columns rawwgt and rltypwgt (of table INDXRTYP) to zero. All of the code-4 and code-6 index names are retrieved from table INDXBINFO and stored in a list. For each index on the list, the following procedure is performed.

All of the component resource categories for that index are retrieved from table INDXGENUS. For each such resource category, Subroutine INDWGT retrieves each actual resource type in that category, along with the total number of resources of that actual type in that category. This latter quantity is stored in column rawwgt of table GENUSTYWGT, in the row where:

- the entry in column genusnam equals the given resource category name being considered;
- the entry in column rltyp equals the given actual type being considered; and
- the entry in column ntlindval equals 0 if NAVMOD models more than one type of the resource, or 1 if NAVMOD models only one type of the resource.

The idea is to use all of the actual resources of the given type in the category regardless of the notional resource type to which they might have been assigned. This quantity is added to the value of column rawwgt of table INDXRTYP in the row for the particular index and actual resource type under consideration. After all resource categories for the index have been processed, column rawwgt for that row will contain the total number of resources of that actual type associated with any resource category and/or notional type related to that index.

Column rawwgt (of table INDXRTYP) having been computed for rows corresponding to the index currently being processed, Subroutine INDWGT then computes normalized weights by setting each entry of column rltypwgt (for rows corresponding to that index) equal to the entry in column rawwgt divided by the sum of the entries in column

rawwgt (this sum being taken over the rows corresponding to that index). If this sum is zero, each normalized weight is set to the reciprocal of the cardinality of the union set. This completes the processing for a given index. The next index on the list is then similarly processed, and so forth.

G. SUBROUTINE EFFVAR ("effectiveness variables")

The preceding subroutines have applied to the first part of the aggregator preprocessor: the determination of the limit, resource, and ordnance stock variables, and the computation of weights for use in aggregation of the effectiveness variables. Subroutines EFFVAR and AGGEFF make up the second part of the aggregator: the computation of notional values for the effectiveness variables. Subroutine AGGEFF, which performs the actual aggregation, considers one effectiveness variable at a time. Subroutine EFFVAR reads values from the input options guide file and determines which effectiveness variables the user wants to have processed in the current run of the program. For each such variable, Subroutine EFFVAR ensures that the appropriate more disaggregated data values are in the INGRES database and then calls Subroutine AGGEFF to compute the notional values.

Subroutine EFFVAR starts by opening an output file, named EFFVARS.AGD. The notional values will be written onto this file in a format readable by NAVMOD.

The format for the input options guide file is indicated in Chapter I, Section C.5. Subroutine EFFVAR receives, as a passed parameter, an option code number that was read by the main program. The action taken by Subroutine EFFVAR depends on this code number. (Here we describe the treatment for option codes 1, 2, and 3. These are considered, in some sense, to be the standard options. Some additional options are discussed in Chapter IV, Section A.)

If the option code is 1, Subroutine EFFVAR retrieves all of the distinct file names (with extensions) from INGRES table EVARINFO and stores them in a working array. If the option code is 2, Subroutine EFFVAR reads all of the file names listed in the input options guide file and stores them. If the option code is 3, Subroutine EFFVAR reads all of the variable names listed in the input options guide file and stores them.

For option codes 1 and 2, the following procedure is performed for each file name in the stored list, in turn. The device and directory specification is retrieved from INGRES table PFNAMEF and joined with the file name to produce a full file specification. (All of

the more disaggregated effectiveness data values files should reside in the directory specified in table PFNAMEF.) The old contents of table RDATEFF, if any, are deleted, and the information in the specified data values file is read into table RDATEFF. Table RDATEFF is then modified to an ISAM storage structure to speed up retrieval. (See Chapter IV, Section A, for some implications of this.) The names of the effectiveness variables with data in this file are retrieved from INGRES table EVARINFO. For each such variable, Subroutine EFFVAR calls Subroutine AGGEFF to compute notional (aggregated) values for that variable.

For option 3, Subroutine EFFVAR uses the following procedure for each variable name on its list, in turn. First, it checks to see that the variable, as named, is actually an input to NAVMOD (to guard against misspelling and other errors); if not, an error message is printed and the variable is ignored. Then, the data values file for that variable is determined by a look-up from table EVARINFO. INGRES table EFFTABCUR contains the name of the data values file whose contents currently also reside in INGRES table RDATEFF. (There is at most one such file.) If this file is the same as the one for the variable to be processed, Subroutine EFFVAR merely calls Subroutine AGGEFF, for that variable. Otherwise, table RDATEFF is cleared out, the data values from the file for the variable to be processed are read into table RDATEFF, table RDATEFF is modified to an ISAM storage structure, and table EFFTABCUR is updated to the name of the new data values file. Then, Subroutine EFFVAR calls Subroutine AGGEFF for this variable.

After all variables have been processed, Subroutine EFFVAR closes the output file EFFVARS.AGD, which now contains data lines for the aggregated values. The aggregator program then ends.

H. SUBROUTINE AGGEFF ("aggregate effectiveness variables")

Subroutine AGGEFF is in some sense the most important routine of the aggregator preprocessor. It averages the more disaggregated data values for an effectiveness variable into data values suitable for use by NAVMOD. Subroutine AGGEFF is called repeatedly by Subroutine EFFVAR. Each call treats one effectiveness variable, but if the variable (as used in NAVMOD) is an array, the call computes an aggregated data value for each element of the array. The name of the variable is passed as a parameter into Subroutine AGGEFF. (See Section G for a description of which variables Subroutine EFFVAR asks Subroutine AGGEFF to process.)

Given a variable name, Subroutine AGGEFF starts by retrieving from INGRES table RVARINFO certain information about the variable, as shown in Table III-6. As indicated in Table III-6, the information is assigned to certain program variables. If the variable name is not found in INGRES table RVARINFO (perhaps due to a misspelling) or if the variable code (column vcoder) of this variable is not equal to 1, indicating that this is not an effectiveness variable, then an informative message is printed and the routine ends. Otherwise, certain information about each dimension index is retrieved from INGRES table INDXBINFO and stored in certain program variables, as indicated in Table III-7.

The rest of Subroutine AGGEFF is divided into two broad parts. The first part retrieves certain sets of weighting factors (weights) from several different INGRES tables, where they were stored after being computed in the first part of the aggregator preprocessor. Appropriate names of resource types are also retrieved. The particular sets of weights and resource names retrieved depend on the nature of the dimension indices of the effectiveness variable being considered, as will be discussed in Section 3. The second part of Subroutine AGGEFF retrieves the more disaggregated data values and, in conjunction with the weights, computes aggregated (weighted average) data values suitable for use by NAVMOD. To aid the reader in understanding the algorithms of Subroutine AGGEFF, the remainder of this section is organized as follows. Section 1 describes, in algebraic notation, the actual weighted averaging procedure or central aggregation algorithm. Each use of this procedure produces a value for exactly one array element of the variable (as used in NAVMOD). If the variable is a scalar, the central aggregation algorithm is performed once. Otherwise, a sequence of these weighted averagings is performed, one for each array element that will be used by NAVMOD. Some program variables related to this sequencing are discussed in Section 2. Finally, Section 3 discusses the retrieval of the weights and resource types.

As averaged values are computed, they are stored in a COMMON block. After averaged values for all relevant array elements for the variable under consideration have been computed, Subroutine AGGEFF calls Subroutine GETALL, which writes these values onto the file EFFVARS.AGD, in a format readable by NAVMOD.

1. Central Aggregation Algorithm

The central aggregation algorithm itself is a straightforward weighted averaging procedure. It computes a value for one NAVMOD input effectiveness scalar variable or for one element of a NAVMOD input effectiveness array variable.

Table III-6. INFORMATION RETRIEVED FROM INGRES TABLE RVARINFO BY SUBROUTINE AGGEFF

Column of RVARINFO	Brief Description[a]	Associated FORTRAN Variable
vnamer	Variable name	NAME
vcoder	Variable code number	IVCOD
nodr	Total number of dimensions (NAVMOD plus added)	NODR
nodn	Number of NAVMOD dimensions	NODN
nfn	1 if variable is integer; 2 if real	NFN
indx1	Name of first index	INDX(1)
indx2	Name of second index	INDX(2)
indx3	Name of third index	INDX(3)
indx4	Name of fourth index	INDX(4)
indx5	(Not used)	INDX(5)
indx6	(Not used)	INDX(6)

[a]For more complete descriptions of the columns of table RVARINFO, see Chapter II, Section C.1.b.

Table III-7. INFORMATION RETRIEVED FROM INGRES TABLE INDXBINFO BY SUBROUTINE AGGEFF

Column of INDXBINFO	Brief Description[a]	Associated FORTRAN Variable[b]
indxnam	Name of Jth index of variable under consideration	INDX(J)
indxcode	Index code number	IXCOD(J)
nsteps	Number of steps for this index	NSTEPV(J)
ivalrl	Number of actual resource types associated with this index	NVALRL(J)
ivalntl	Number of notional resource types or numerical upper limit	NVALNT(J)
genus	Single resource category, if any, associated with this index	GENUS0(J)

[a] For more complete descriptions of the columns of table INDXBINFO, see Chapter II, Section B.3.a.
[b] "J" represents the J-th index or dimension of the variable under consideration.

Let D_1 be the number of dimensions of the variable as it is used in NAVMOD, let D_2 be the total number of dimensions of the variable as it is input to the aggregator and let D_3 be the upper bound on dimensioning set by the program.[11,12] It is assumed that $D_1 \leq D_2 \leq D_3$. For each value d between 1 and D_2, inclusive, ordered sets Ω_d of real numbers (weights) and C_d of component names (types) have been determined previously in the program. The sets Ω_d and C_d are structured to have the same cardinality, denoted by n_d. Let ω_{id} denote the i^{th} element of set Ω_d and c_{id} the i^{th} element of set C_d. (For d such that $D_2 < d \leq D_3$, $n_d = 1$, $\omega_{1d} = 1.0$, and c_{1d} is the blank character.)

Consider a D_3-tuple $\left(i_1, i_2, \ldots, i_{D_3}\right)$ of integers such that $1 \leq i_d \leq n_d$ for all d, $1 \leq d \leq D_3$. Let c denote the D_3-tuple of types

$$c = \left(c_{i_1 1}, c_{i_2 2}, \ldots, c_{i_{D_3} D_3}\right);$$

c is an element of the cartesian product

$$C = C_1 \times C_2 \times \ldots \times C_{D_3}.$$

There is a one-to-one correspondence between elements of C and tuples $(i_1, i_2, \ldots, i_{D_3})$.

Define, for each tuple c,

$$\theta_c = \prod_{d=1}^{D_3} \omega_{i_d d}.$$

The program has been structured so that for each d,

$$\sum_{i=1}^{n_d} \omega_{id} = 1.$$

It is straightforward to prove then that

---

[11] In the currrent program, D_3=6, and the computer code can handle values of $D_1 \leq 3$. Some reprogramming would be necessary to increase these limits. It was desired that INGRES table RDATEFF and the flat files that contain the data values be no more than 80 columns wide, to facilitate display and editing. With the current formats, this restricts D_2 to values less than or equal to 4. If more than 80 columns are allowed, it would be relatively easy to change the table and file formats to allow values of D_2 less than or equal to 6.

[12] The FORTRAN variables for D_1 and D_2 are NODN and NODR, respectively; see Table III-6.

$$\sum_{c \in C} \theta_c = 1.$$

Let v_c denote the input value to the aggregator for the array element corresponding to tuple c for the variable currently being treated. The v_c are stored in INGRES table RDATEFF. (The reader might wish to review the discussion in Chapter II, Section D.) Define the subset M of C as the set of $c \in C$ such that $v_c < 0$ or c does not appear in the file or INGRES table of data values for the variable. This coding convention allows the aggregation algorithm to ignore certain combinations of resources, as will be explained soon. Let

$$W = \sum_{c \in M} \theta_c .$$

The final result of the central aggregation algorithm, i.e., the aggregated value used as the value of the NAVMOD input scalar variable or array element is computed as

$$\bar{v} = \begin{cases} \left(\sum_{c \in C \setminus M} \theta_c v_c \right) \Big/ (1-W) & W<1, \\ 0 & W=1. \end{cases}$$

(The notation $C \setminus M$ means the difference of the sets C and M.) The calculation of W and $\bar{v}$ is done in double precision. The value $\bar{v}$ is converted to single precision and is stored in the appropriate element of array ZVALU for later output. (The appropriate element is determined by Subroutine PUTONE.) Note that $\bar{v}$ is a true weighted average (i.e., a convex combination) of the elements of the set

$$\{v_c | c \in C \setminus M\}.$$

The construction with M and W allows the user to remove from consideration combinations of types of resources that would not in fact occur. An example of the use of this construction was presented in Chapter I, Section B.1.b (variable ABPDS). As an additional example, consider a variable related to the effectiveness of Blue sea-based fighter-aircraft; in NAVMOD, it is a scalar (NAVMOD models only one notional type of such aircraft), but the inputs to the aggregator have two dimensions--a Blue fighter type and a munition type. Assume (for purposes of this example) that the sets of actual resource types being considered are {F-14, F/A-18} and {Phoenix, Sparrow}, for these two

dimensions, respectively. The aggregator accepts values v_c for all four aircraft-type/munition-type combinations c. Given that the combination of F/A-18 and Phoenix does not actually occur, the user can: 1) input v_c for this combination as a negative number (or delete it from the file of data values, which has the same effect), or 2) use a value of 0 for this combination. The effects of these two options are different. In the first case, the negative value will be ignored and the aggregated value would be a weighted average of the effectiveness values of the three feasible aircraft-type/munition-type combinations. In the second case, the effectiveness values of all four combinations would be averaged, (assuming that all weights θ_c are positive) and the effectiveness value of zero for the F/A-18/Phoenix combination would yield a lower aggregated value. (This lower value might be interpreted as taking into account the possibility of a mismatch between availaole platforms and munition types.) By suitably setting the v_c either of these options can be used as desired. (That is, the use of negative values for v_c is a convention, instructing the aggregation algorithm to ignore such combinations c.)

Table III-8 shows the correspondence between some of the preceding algebraic notation and the FORTRAN notation in the computer program. Some further remarks on the computer implementation of the central aggregation algorithm are as follows.

Given D3-tuple of component names $c = \left(c_{i_1 1}, c_{i_2 2}, \ldots, c_{i_{D_3}, D_3}\right)$ the program finds the data value v_c by retrieving from INGRES table RDATEFF the value in column val in the row where

- the entry in column vnamer equals the name of the variable under consideration;
- the entry in column comp1 equals the component name $c_{i_1 1}$; and
- the entry in column compd equals the component name $c_{i_d d}$, for all values of d from 2 through 6 (D3), inclusive.

The normal situation is for there to be exactly one such row. If there is no such row, the tuple is not included in the aggregation, as discussed above. If, by some chance, there are two or more data lines in RDATEFF with the same variable name and sequence of component names, an INGRES error occurs and program execution stops. This situation most likely will have been detected at earlier stages of program operation.

Table III-8. CORRESPONDENCE OF ALGEBRAIC AND COMPUTER NOTATION IN THE CENTRAL AGGREGATION ALGORITHM

Algebraic Variable	FORTRAN Variable	Mnemonic
d	J (or actual numerical values)	
D_1	NODN	(number of dimensions--NAVMOD)
D_2	NODR	(number of dimensions--real)
D_3	(fixed at 6)	
ω_{id}	WGTU(IA, J)	(weights used)
c_{id}	TYPU(IA, J)	(types used)
n_d	LIMU(J)	(limits used)
$i_1, i_2..$	IA1, IA2,...	
$c_{i_d d}$	RLTYP(d)	(real type)
θ_c	WPROD	(weights product)
v_c	DVAL	(double precision value)
W	WASTSM	(waste sum)
$\bar{v}$	VALFIN	(final value)

In the computer code of Subroutine AGGEFF, the central aggregation is implemented by means of a set of six nested DO loops. (Six is the value of D_3, and as the program is currently written, represents an upper bound on the total number of dimensions, NAVMOD plus additional, that a variable can have. Increasing this limit would involve adding a new DO loop to the nest or rewriting the algorithm completely.) The d^{th} loop corresponds to the d^{th} dimension of the variable. The labels of these loops are 210, 220, 230, 240, 250, and 260, respectively, and the lower and upper limits for the d^{th} loop are given by the working variables LIML(d) and LIMU(d). All LIML(d) are set equal to 1 and the LIMU(d) correspond to the values n_d discussed earlier. For d such that $D_2 < d \leq D_3$, LIMU(d) is set to 1. The multiplication of the weights ω_{id} and the use of elements of the cartesian product $C = C_1 \times C_2 \times \ldots \times C_{D_3}$ is accomplished through the nested loops.

Suppose that (using the notation introduced earlier in this subsection) some weight ω_{id} is zero. Then for any $c \in C$ such that the d^{th} component of c equals i, the overall weight θ_c is zero, and thus the value v_c is not counted in the aggregation, and therefore need not be retrieved from a data table. The DO loop structure takes advantage of this fact as appropriate, as can be seen from the code. It is certainly possible that for many databases, a large proportion of the ω_{id} will in fact be zero.

2. Sequencing the Central Aggregation Algorithm

To compute aggregated values for all elements of a NAVMOD input array variable, the central aggregation algorithm is performed once for each array element. Between each performance, the sets C_d and/or Ω_d are (in general) reset for some values of d. That is, the sets C_d and Ω_d can vary with the particular NAVMOD array element it is desired to compute a value for, as described in this subsection.

Assume for now that $D_1 \geq 1$; the case $D_1 = 0$ will be discussed presently. For $d = 1,\ldots,D_1$, let ι_d be the number of elements along the d^{th} dimension that it is desired to use in NAVMOD. Depending on the type of index, the quantity ι_d might be the value of a limit variable, the value of some constant (numerical or symbolic) or the sum of some combination of limit variables and/or constants. The values ι_d are stored in column ivalntl of INGRES table INDXBINFO. Each combination $\left(j_1,j_2,\ldots,j_{D_1}\right)$ where $1 \leq j_d \leq \iota_d$ for each d corresponds to a different array element of the variable as it will be used in NAVMOD. For each such combination, sets C_d and Ω_d and cardinalities $n_d = |C_d| = |\Omega_d|$ are established for $d = 1,\ldots,D_2$, and the central aggregation algorithm is performed. With one exception (numerical indices, to be discussed presently) the C_d, Ω_d, and n_d are extracted from arrays that have themselves previously been determined via extraction of information from INGRES tables, as discussed in Section 3. For now, we focus on the construction of the Ω_d, C_d, and n_d. Assume that we are trying to obtain a value for element $\left(j_1,j_2,\ldots,j_{D_1}\right)$ of the variable as it is used in NAVMOD.

For values of d between 1 and D_1, inclusive, these sets are determined as follows. If the d^{th} dimension index of the variable is a numerical index, then the algorithm sets $n_d=1$, $\omega_{1d} = 1.0$, and c_{1d} to a character encoding of the integer j_d (so that essentially, 1 is encoded as '01', 2 as '02',...,10 as '10', and so forth; see Section I.1 for more information). Otherwise, if the d^{th} dimension is a resource-based index, then the algorithm sets

$$n_d = \text{LIMAR1}(j_d,d)$$

$$\omega_{id} = \text{WGTAR1}(i,j_d,d) \quad i = 1,\ldots,n_d, \text{ and}$$

$$c_{id} = \text{TYPAR1}(i,j_d,d) \quad i = 1,\ldots,n_d.$$

The contents of the FORTRAN arrays LIMAR1, WGTAR1, and TYPAR1 have been determined earlier in the subroutine, as described in Section 3. (LIMAR1 is an integer array, WGTAR1 is real, and TYPAR1 is character-length-10. The mixture of FORTRAN and algebraic notation should cause no confusion.) The weights WGTAR1 have been normalized, so that

$$\sum_{i=1}^{n_d} \mathrm{WGTAR1}(i,j_d,d) = 1,$$

for all appropriate j_d and d.

Note that (for $1 \le d \le D_1$) the weights, types, and limits can vary with the particular NAVMOD array element $\left(j_1,j_2,\ldots,j_{D_1}\right)$ for which a value is being computed. The weights WGTAR1(i,j_d,d) can in general be different for each different value of j_d. The limits and types will be the same, however, for any values j_d that point to the same resource category, as explained in Section 3.

For added dimensions, i.e., values of d between $D_1 + 1$ and D_2, inclusive, the code sets

$$n_d = \mathrm{LIMAR2}(s,d-D_1)$$

$$\omega_{id} = \mathrm{WGTAR2}(i,s,d-D_1) \qquad i = 1,\ldots,n_d, \text{ and}$$

$$c_{id} = \mathrm{TYPAR2}(i,s,d-D_1) \qquad i = 1,\ldots,n_d .$$

The arrays LIMAR2, WGTAR2, and TYPAR2 are to added dimensions what LIMAR1, WGTAR1, and TYPAR1 are to NAVMOD dimensions. Their contents have been determined earlier in the subroutine, as discussed in Section 3. The weights WGTAR2 satisfy the normalization condition

$$\sum_{i=1}^{n_d} \mathrm{WGTAR2}(i,s,d-D_1) = 1,$$

for all appropriate s and d. The subscript s is equal to unity unless the d-th index of the more disaggregated variable is an adaptive index, as discussed in Chapter II, Section E. The treatment of adaptive indices is discussed in Section 3.b.4; for now, assume that the variable does not have an adaptive index. In this latter case, the arrays LIMAR2, WGTAR2, and TYPAR2 do not depend on the values j_d (i.e., on the NAVMOD array

element being considered), and thus for d such that $D_1 < d \leq D_2$, the sets C_d and Ω_d, and their cardinality n_d, do not depend on those values.

Finally, for d between $D_2 + 1$ and D_3, inclusive, the code sets $n_d = 1$, $\omega_{1d} = 1.0$, and c_{1d} to the blank character.

Some remarks about the cases $D_1 = 0$, $D_1 = D_2$, and $D_1 = D_2 = 0$ are in order (recall that $D_1 \leq D_2$, always). If $D_1 > 0$ and $D_2 = D_1$, so that the more disaggregated variable has no added dimensions, then the procedure described for $D_1 + 1 \leq d \leq D_2$ is simply not performed. If $D_1 = 0$, but $D_2 > 0$, (so that the NAVMOD variable is a scalar, but some dimensions have been added for the more disaggregated variable), then the central aggregation algorithm is performed once, with the sets C_d and Ω_d and cardinalities n_d being defined from WGTAR2, TYPAR2, and LIMAR2, as indicated above, with s=1 and $1 \leq d \leq D_2$.

Finally, if $D_1 = D_2 = 0$, then both the NAVMOD variable and the more disaggregated variable are scalars, and there is (at most) one data line and data value for the variable. In this case, the weight θ_c equals unity for the one value of c. The data line is simply retrieved from the more disaggregated data values file and its data value is assigned to the appropriate element of the array (ZVALU) of aggregated values. (If the value is negative, or if the data line is missing, the resultant value is set to zero.)

Table III-9 shows the correspondence between some of the algebraic notation used above and the variables in the computer code. (The reader may also wish to re-examine Table III-8).

In the computer code of Subroutine AGGEFF, the sequencing of the central aggregation algorithm is implemented by means of a set of three nested DO loops (with labels 110, 120, and 130). The loops are sufficient to keep track of three dimensions. None of NAVMOD's inputs are arrays of more than three dimensions; if such arrays were added, additional DO loops would be needed and/or the sequencing algorithm would have to be rewritten in some manner. The lower and upper limits of the Jth loop are given by the working variables LIMOL(J) and LIMOU(J), respectively. These variables are set in Subroutine AGGEFF, as follows. For each J (if any) such that NODN < J ≤ 3 (i.e., $D_1 < J \leq 3$), LIMOL(J) and LIMOU(J) are both set to zero (and the Jth loop is performed once, with its index set to zero, for each setting of the loops encasing it). For each J such that 1 ≤ J ≤ NODN, (i.e., $1 \leq J \leq D_1$), LIMOL(J) is set to 1 and LIMOU(J) is set to the value

Table III-9. ALGEBRAIC AND COMPUTER NOTATION USED IN THE SEQUENCING OF AGGREGATIONS

Algebraic Variable	FORTRAN Variable	Mnemonic
D_1	NODN	number of dimensions--NAVMOD
D_2	NODR	number of dimensions--real
j_1, j_2, j_3	IN1, IN2, IN3	
l_d	LIMOU(d)	outer upper limit
i	IA	
s	IST	indicator for step (used for adaptive indices)
d	J (appropriate numerical value)	
$d-D_1$	JD	
	LIMAR1(INd,d)	first limit array
	WGTAR1(IA, INd, d)	first weight array
	TYPAR1(IA, INd, d)	first type array
	LIMAR2(IST, JD)	second limit array
	WGTAR2 (IA, IST, JD)	second weight array
	TYPAR2 (IA, IST, JD)	second type array

NVALNT(J) as indicated in Table III-7. This value, retrieved from table INDXBINFO,was either preset in the database or computed by Subroutine LMVSET or LMVCMP in the first part of the preprocessor. In the special case NODN=0 (i.e., the NAVMOD variable is a scalar) all elements of LIMOL and LIMOU are set to zero.

The set of six nested DO loops that form the central aggregation is itself nested within the nest of three DO loops for the sequencing algorithm, thus overall, Subroutine AGGEFF contains a set of nine nested DO loops. However, $(3-D_1)+(6-D_2)$ (i.e., (3–NODN)+(6–NODR)) of these loops will be performed only once.

3. Retrieval and Storage of Resource Types and Weights

The arrays of types, weights, and limits TYPAR1, TYPAR2, WGTAR1, WGTAR2, LIMAR1, LIMAR2, and LIMOU referred to in the previous section are filled with values in the first part of Subroutine AGGEFF, before the sequence of aggregation

algorithms begins. These values have been computed by the first part of the aggregator preprocessor program and are stored in a number of different INGRES tables. This section describes the transfer of data from those INGRES tables to these arrays. With the exception of adaptive indices (the treatment of which is described in subsection b.4), a separate transfer is performed, in turn, for each dimension index. The treatment is somewhat different for dimension indices that are used in NAVMOD (i.e., d such that $1 \leq d \leq D_1$) as opposed to dimension indices added to the variable in forming the more disaggregated data (d such that $D_1 + 1 \leq d \leq D_2$), as discussed below. Table III-10 summarizes the discussion of the following subsections. (The reader also might wish to review the notation in Tables III-6 through III-9.)

a. Treatment of Dimension Indices Used in NAVMOD

(1) General Remarks

For dimensions d such that $1 \leq d \leq D_1$, there are three separate cases.

If the index involved is a numerical index (which for dimensions d such that $1 \leq d \leq D_1$ corresponds to an index code greater than or equal to 10), no retrievals from INGRES tables are performed; the index is treated as described in Section 2.

Otherwise, the index is assumed to be resource based. The program perfonns some tests to determine if certain limits specified for this index exceed the current dimensioning of certain working variables in the subroutine. In this case, an informative message is printed (on the file associated with logical unit 4) and the subroutine ends. Otherwise, the program separates the cases: single resource category versus compound resource category.

(2) Single Resource Category Indices

The working variable NSTEPV(d) gives the number of resource categories or steps associated with the d^{th} index (see Table III-7). If NSTEPV(d) equals unity (and the index is resource based), the d^{th} index is associated with a single resource category, and the name of this category has been retrieved from column genus of INGRES table INDXBINFO and stored in the working variable GENUS0(J). Information on the actual resource types occurring in this category, and the (normalized) weights with which these types should be considered, is stored in the INGRES table GENUSTYWGT. (The resource types were preset in table GENUSTYWGT before the preprocessor started; the weights were computed in the first part of the aggregator preprocessor, in Subroutine RSOWGT.)

Table III-10. WHERE THE COMPONENT NAMES AND AGGREGATION WEIGHTS ARE RETRIEVED

Type of Index	If NAVMOD Index	If Added Index
Numerical	Use one character-encoded integer, with a weight of 1.0	n/a
Single Resource Category	Retrieve information from table GENUSTYWGT, for appropriate notional type	Retrieve information from table GENUSTYWGT, for total over all notional types
Compound Resource-based	Retrieve information from table GENUSTYWGT, for each appropriate notional type within each appropriate resource category	Retrieve information from table INDXRTYP
"Detail" (index code 30)	n/a	Retrieve information from table INDXDTYP
"Adaptive" (index code 10)	n/a	Retrieve information from table GENUSTYWGT, for total over all notional types, for each step of index

INGRES table GENUSTYWGT contains a different set of weights for each possible (numerical integer) value j_d of the index as it is used in NAVMOD. That is, each type-j_d NAVMOD resource (for example, type-2 escort ships) in this category can be associated with a different set of weighting factors over the actual resource types in the resource category (e.g., the set of actual types of Blue escort ships being considered). The quantity j_d varies from 1 through a limit variable ι_d which was retrieved from column ivalntl of INGRES table INDXBINFO (where it was either preset or computed in Subroutine LMVSET or Subroutine LMVCMP). Let i index the actual resource types associated with the resource category. Consider the row of INGRES table GENUSTYWGT where

- the entry in column genusnam equals the name of the resource category; and
- the entry in column ntlindval equals the desired value of j_d (integer); and
- the entry in column ordno equals the desired value of i (integer).

There should be exactly one such row; if not, an error message is printed and the program terminates. Otherwise

- the working variable TYPAR1(i, j_d, d) is set to the value in column rltyp on this row (a character string of length 10); and

- the working variable WGTAR1(i, j_d, d) is set to the value in column rltypwgt on this row (a real number).

Note that in the case of a single resource category, TYPAR1(i, j_d, d) does not vary for different j_d; that is, the list of actual resource types associated with this index does not depend on the NAVMOD array element into which the effectiveness values will be aggregated. For ease in data retrieval, the set of types has been repeated in table GENUSTYWGT for each value of j_d. The set of weights associated with this index does vary with j_d, as indicated above. For all j_d, working variable LIMAR1(j_d, d) is set equal to the number of actual resource types associated with the resource category. In the case where the index corresponds to a single resource category, this value does not depend on j_d. The value has been retrieved from column ivalrl of table INDXBINFO, as indicated in Table III-7.

(3) Compound Resource-based Indices

If the dth index is associated with a sequence of resource categories, the treatment is somewhat more complicated. First, the program retrieves certain information about this sequence of categories from the INGRES table INDXGENUS, as shown in Table III-11. The database has been structured so that: 1) the entry in column nsteps of INGRES table INDXBINFO for the dth index, which has been assigned to the variable NSTEPV(d) and is also denoted here by the algebraic symbol S, is equal to the number of resource categories in the sequence, and, 2) the sum of the numbers of NAVMOD elements associated with these categories is equal to the total number of NAVMOD elements actually used along this dimension, i.e.,

$$\iota_d = \sum_{s=1}^{S} m_{sd} ,$$

where m_{sd} is as defined in Table III-11. Let us define, for each a=1,...,S, the partial sum

$$\overline{m}_{ad} = \sum_{s=1}^{a} m_{sd},$$

and define $\overline{m}_{0d}$ as zero. (The $\overline{m}_{ad}$ have no exact equivalent in the computer code but help convey the essence of the algorithm.) The FORTRAN (integer) working array ISTEPV is defined by

Table III-11. INFORMATION RETRIEVED FROM INGRES TABLE INDXGENUS FOR COMPOUND RESOURCE-BASED INDICES

Column of Table INDXGENUS	Brief Description	Associated FORTRAN Variable[a]	Algebraic Notation
indxnam	Name (of compound resource-based index)	INDX(J)	----
istep	Step number (1,2,...)	IST	s
stepgenus	Resource category associated with this step (CHARACTER*6)	GENU1(IST,J)	---
steplvval	Number of notional types of this resource	LVVALV(IST)	m_{sd}

[a]Here, J is the same as d, the order number of the dimension being considered.

$$\text{ISTEPV(k,d)} = \text{s if and only if } \overline{m}_{s-1,d} < k \le \overline{m}_{sd} .$$

ISTEPV is saved for later possible use with adaptive indices, if any, as described in Section 3.b.4). The interpretation is that the k^{th} element of the d^{th} dimension of the NAVMOD variable is associated with the s^{th} (i.e., the ISTEPV(k,d)–th) resource category.

The following procedures are performed for each resource category in the sequence, in turn. First, the number of actual (not notional) resource types associated with the category is retrieved from INGRES table GENUSINFO and is stored in the working variable NRTY (mnemonic for number of real types). If this is the s^{th} category in the sequence, the assignment

$$\text{LIMAR1(k,d)} = \text{NRTY if and only if } \overline{m}_{s-1,d} < k \le \overline{m}_{sd}$$

is made.

Then, a series of retrievals from the INGRES table GENUSTYWGT is performed, which is similar to the single resource category case. For i = 1 through NRTY and for j = 1 through m_{sd} there should be exactly one row of INGRES table GENUSTYWGT where

- the entry in column genusnam (of table GENUSTYWGT) is the name (a character string of length 6) of the resource category under consideration;
- the entry in column ntlindval equals the value of j; and
- the entry in column ordno equals the value of i.

(If this is not the case, an error message is printed and the program terminates.) Let RLTYPB (a character string of length 10) denote the entry on this row in column rltyp (mnemonic for real type) and let WGTYPB (a real variable) denote the entry on this row in column rltypwgt. Then the assignments are made

$$\mathrm{WGTAR1}(i, \overline{m}_{s-1,d}+j,d) = \mathrm{WGTYPB}, \text{ and}$$

$$\mathrm{TYPAR1}(i, \overline{m}_{s-1,d}+j,d) = \mathrm{RLTYPB}.$$

The interpretation here is that if an aggregated value is being computed for an array element $(e_1, e_2,\dots,e_{D_1})$ of the NAVMOD variable where $e_d = k$, then the set of actual weapon types

$$\{\mathrm{TYPAR1}(i, k-\overline{m}_{s-1,d}, d) \mid i=1,\dots,I_{sd}\}$$

should be used, where the i^{th} element of this set is weighted using the weight

$$\mathrm{WGTAR1}(i, k-\overline{m}_{s-1,d}, d),$$

where s is such that

$$\overline{m}_{s-1,d} < k \leq \overline{m}_{sd}$$

and I_{sd} is the number of actual weapon types associated with the s^{th} resource category in the sequence of resource categories for index d (I_{sd} equals LIMAR1(k,d)).

Note that the $\mathrm{TYPAR1}(i, \overline{m}_{s-1,d}+j, d)$ depend on i but do not depend on j, as long as $1 \leq j \leq m_{sd}$--the set of actual resource types is associated with a resource category, rather than with the specific NAVMOD index value. (TYPAR1 is dimensioned as it is to facilitate computation.) The weights $\mathrm{WGTAR1}(i, \overline{m}_{s-1,d}+j, d)$ in general can depend on both i and j.

b. Treatment of Added Indices

The treatment of indices that the user has added to the variable (for increasing the detail and realism of the more disaggregated data) is somewhat different from that of indices used for the NAVMOD variable, even if the same index name is used. (The aggregator treats an index that is used as an added index differently than an index used in NAVMOD.) Except in the case of adaptive added indices, as discussed in Section 3.b.4), added indices have the following characteristics:

- Exactly one set of actual resource types is associated with the index;
- Exactly one set of (normalized) weights is associated with the index;
- These types and weights do not vary with the particular element of a NAVMOD array for which the aggregator is computing a value.

The sets of types and weights are stored in several different INGRES tables, depending on certain characteristics of the index. Subroutine AGGEFF retrieves these types and weights into the arrays TYPAR2 and WGTAR2, respectively, and the number of types (and weights) is retrieved into the array LIMAR2, as indicated in Section 2. This section presents the details of this retrieval and explains the different cases.

For effectiveness variables, the employment of added indices is at the discretion of the user of the preprocessor: subject to mild limitations, the user can decide which variables are to have added indices, how many added indices, and which indices these should be.

Throughout this subsection, let J or d denote the order number of the dimension of the variable, as before, and let JD = J − NODN = $d - D_1$ denote the order number of the added dimension. The weight and type retrieval procedures of this section are performed for each value of d from D_1+1 through D_2 (in FORTRAN notation, NODN+1 through NODR), inclusive. The different cases distinguished by the program are as follows.

(1) Detail Indices

This case is for indices that add a level of detail that is finer than NAVMOD's resources. As discused in Chapter II, Section B.2, and Chapter III, Section B, such indices are characterized by a code number of 30 in column indxcode of INGRES table INDXBINFO. The different actual resource types (or other salient characteristics) and their

associated weights are stored in INGRES table INDXDTYP. The user preset the list of resource types and raw weights in this table; Subroutine STINDD has computed normalized weights from the raw weights. Subroutine STINDD also computed the number of actual resource types (which can be denoted n_d) associated with the index and stored it in column ivalrl of INGRES table INDXBINFO; earlier in Subroutine AGGEFF, this value was retrieved from there (see Table III-7), and it is now assigned to the array element LIMAR2(1,JD). For each value of i from 1 through n_d, the row of INGRES table INDXDTYP is found where the entry in column indxnam equals the name of the index under consideration and the entry in column ordno equals the value of i. There should be exactly one such row; if not, an error message is printed and the program terminates. Otherwise:

- the value in this row in column rltyp (a character string of length 10) is retrieved and assigned to the (character) array element TYPAR2(i,1,JD), and
- the value in this row in column rltypwgt (a real number) is retrieved and assigned to the array element WGTAR2(i,1,JD).

(2) Single Resource Category Indices

If the index is not a detail index or an adaptive index, and the variable NSTEPV(d) equals 1 (recall that NSTEPV(d) was retrieved from column nsteps of INGRES table INDXBINFO; see Table III-7), then the index corresponds to a single resource category. The name of this category was retrieved from column genus of INGRES table INDXBINFO. (Recall that a numerical index cannot be used as an added index (see the discussion in Chapter II, Section B.2), as in that case, no weights for averaging can be formed.)

The actual resource types and weights associated with the index are retrieved from INGRES table GENUSTYWGT, but the procedure differs from that described in Section 3.a.2). In that section, the set of weights could vary with the NAVMOD type of resource being fitted. Here, a different, overall set of weights is used, based on the total numbers of actual resource types (for example, see Sections B.3.d and B.3.e.). The number of actual resource types is given by the variable NVALRL(J), or n_d, which is assigned to the array element LIMAR2(1,JD). The working variable NTLVAL is defined as 0 if NAVMOD can model multiple types of resources within this resource category; 1, if not. (The program obtains this information from INGRES table LIMVINFO.) For values of i from 1 through n_d, there should be exactly one row of INGRES table GENUSTYWGT where

- the entry in column genusnam equals the name of the resource category under consideration (a character string of length 6);
- the entry in column ntlindval equals the value of variable NTLVAL; and
- the entry in column ordno equals the value of i.

If this is not the case, an informative message is written and the program ends. Otherwise

- TYPAR2(i,1,JD) is assigned the entry in column rltyp of this row (a character string of length 10); and
- WGTAR2(i,1,JD) is assigned the entry in column rltypwgt of this row (a real number).

(3) Compound Resource-based Indices

The d-th added index might be associated with a set of two or more resource categories. This case occurs if the index is not a detail or adaptive index and NSTEPV(d) ≥ 2. The name of such an index would be the same as an index considered in Section 3.a.3), but the treatment for added indices differs from that of indices used by the NAVMOD variable. Rather than pointing to a sequence of resource categories, the index is considered here as encompassing a set of actual resource types, determined as the union of all the resource types in the resource categories associated with the index. This union set of types was preset before the preprocessor was run and the information is stored in INGRES table INDXRTYP. The set is ordered; each resource type has an associated order number. The number of elements in this set has been assigned to the variable NVALRL(J) (see Table III-7), or n_d, whence it is assigned to LIMAR2(1,JD). For each resource type, an associated weight has been computed by Subroutine INDWGT in the first part of the preprocessor, as described in Section F, and has been stored in INGRES table INDXRTYP. For each value of i from 1 through n_d, there should be exactly one row of table INDXRTYP where

- the entry in column indxnam equals the name of the index under consideration (a character string of length 6), and
- the entry in column ordno equals the value of i.

If there is not exactly one such row, then an informative message is printed, and the program terminates. Otherwise

- TYPAR2(i,1,JD) is set equal to the entry in column rltyp for this row (a character string of length 10); and

- WGTAR2(i,1,JD) is set equal to the entry in column rltypwgt for this row (a real value).

(4) Adaptive Indices

The procedure used when an added index is an adaptive index is somewhat different from the procedure used for other types of indices. (A review of Chapter II, Section E, might be helpful to the reader.) Let the adaptive index be the d-th index in the sequence of indices for the variable under consideration. The program finds d* such that the d*-th index in the sequence is the base index associated with the adaptive index. Recall that there is at most one adaptive index for a variable, it must have an associated base index, and this base index must be a NAVMOD index for this variable, so that $1 \leq d^* \leq D_1$. The number of steps associated with the base index (and hence, the adaptive index) has already been retrieved from INGRES table INDXBINFO (column nsteps; see Table III-7.)

For each step, the resource category associated with this step (for the adaptive index) is retrieved from table SPLITINDX. A working variable NTLVAL is defined as 0 if NAVMOD can model multiple (notional) types of resources within this resource category, 1, if not (this is similar to the definition of NTLVAL discussed in Section b.2). The number of actual resource types associated with this resource category is retrieved from table GENUSINFO. If this is the s-th step, this number of actual types is assigned to the working variable LIMAR2(s,JD); here, let it be denoted by m_s. The actual resource types and weights associated with this resource category are retrieved from table GENUSTYWGT. For each value of i from 1 through m_s, there should be exactly one row of table GENUSTYWGT where:

- the entry in column genusnam equals the name of the resource category under consideration;
- the entry in column ntlindval equals the value of NTLVAL; and
- the entry in column ordno equals the value of i.

If this is not the case, an informative message is printed and the program ends. Otherwise:

- TYPAR2(i,s,JD) is assigned the entry in column rltyp of this row; and
- WGTAR2(i,s,JD) is assigned the entry in column rltypwgt of this row.

The discussion of the sequencing of the central aggregation algorithm in Section 2, did not treat the case of adaptive indices; it is discussed here. In the notation of Section 2,

suppose that an aggregated value for element $(j_1, j_2, \ldots, j_{D_1})$ of the NAVMOD variable is to be computed and that the d^{th} index of the variable is an adaptive added index. Let d* be the position of the associated base index; it is necessary that $1 \leq d^* \leq D_1$. For the base index, the algorithm sets

$$n_{d^*} = \text{LIMAR1}(j_{d^*}, d^*)$$

$$\omega_{id^*} = \text{WGTAR1}(i, j_{d^*}, d^*) \quad i=1,\ldots,n_{d^*}, \text{ and}$$

$$c_{id^*} = \text{TYPAR1}(i, j_{d^*}, d^*) \quad i=1,\ldots,n_{d^*},$$

just as described in Section 2. Based on j_{d^*}, a step value s is computed. If the base index is numerical, s is defined simply by $s=j_{d^*}$. If the base index is a compound resource-based index, then s is computed as

$$s = \text{ISTEPV}(j_{d^*}, d^*),$$

where the working variable ISTEPV has been computed as described in Section 3.a.3). Then for the adaptive index d, the assignments

$$n_d = \text{LIMAR2}(s, d-D_1)$$
$$\omega_{id} = \text{WGTAR2}(i, s, d-D_1) \quad i=1,\ldots,n_d, \text{ and}$$
$$c_{id} = \text{TYPAR2}(i, s, d-D_1) \quad i=1,\ldots,n_d$$

are made. The form of the above equations is the same as in Section 2, but the value of s is the computed step value, which depends on j_{d^*}.

I. ADDITIONAL SUBPROGRAMS

In addition to the major routines of the aggregator, there are a number of additional subprograms (all subroutines, except the Function ICEIL), some of which have been referred to previously. Section 1 describes Subroutine CHRINI (character initialization), which is used in obtaining component names for numerical indices. Subroutine CLRWK (clear working variables) resets to zero several working variables used in Subroutine RSOWGT. The remaining subprograms relate to the storage and output of aggregated values, and are discussed briefly in Section 2.

For additional information, the reader is referred to Table A-1 of Appendix A, which contains a complete list of program segments of the aggregator preprocessor, with mnemonics and assorted information on calls, COMMON blocks, etc.

1. Subroutine CHRINI ("character initialization")

The subroutine CHRINI is called once, by the main program, at its very beginning. The subroutine encodes each integer I between 1 and 99, inclusive, into a character variable CHRINT(I), of length 10. The characters CHRINT(I) are stored in COMMON block /CHRCMN/, for access by other subroutines.

For values of I between 10 and 99, inclusive, the character variable CHRINT(I), of length 10, is defined as: characters 1 and 2 are the tens and units digits, respectively, of I; characters 3 through 10 are blank. For values of I between 1 and 9, inclusive, CHRINT(I) is defined as: character 1 is '0', character 2 is the digit of I, and characters 3 through 10 are blank. The use of a leading zero facilitates comparison of the character strings. INGRES ignores blanks in comparing character strings, so it considers ' 2' and '2 ' to be lexically greater than '10'. INGRES considers the string '02', however, to be lexically less than '10', as is consistent with the relationship between the corresponding integers.

The motivation for this encoding procedure is as follows. The name of each actual resource type is referred to in the aggregator by a character string of length 10 (abbreviation being used as necessary--simple format changes could expand this limit). Resource-based indices are associated with a list of actual resource names; numerical indices, with a set of numbers {1,...,n}, for some n. Since a given variable might have some numerical dimensions and some resource-based dimensions (and some detail dimensions, as discussed in Section B, which are also associated with a list of character length-10 strings), the encoding scheme provides a common format for the associated lists of names for all types of dimension indices. The sequence of names on each line of a more disaggregated data file can then be read and treated by the program as a sequence of character length-10 variables.

This encoding scheme is predicated on the assumption that no numerical index need ever take on a value greater than 99 (e.g., that no more than 99 regions are ever played). If numerical index values greater than 99 were required, it would be straightforward to change Subroutine CHRINI to use a coding scheme with more digits (e.g., '001', '002', etc.). The new scheme would also need to be implemented in the auxiliary computer programs GENEFFDAT and GENROLDAT, which generate the shell files for the more disaggregated data. Chapter IV, Sections C.5 and C.6, discuss these programs.

2. Routines for Storage and Output of Aggregated Values

Most of the routines for storage and output of aggregated values are similar to the NAVMOD input routine, discussed in Chapter III of Reference [2]. The information file used by NAVMOD is identical to the information file used by the preprocessor. One difference from NAVMOD is that NAVMOD stores the values for its inputs in a set of COMMON blocks; the aggregator essentially concatenates this information into a single (blank) COMMON block, which consists of a single large array, ZVALU (which is equivalenced to an array NVALU to handle the storage and writing of integer values). The variables IVARQQ and IVARQ hold information about the storage locations of the NAVMOD input values (i.e., the aggregated values computed by the preprocessor) and other information about the NAVMOD inputs. The meanings of these arrays are shown in Table III-12, which is virtually identical to Table III-6 of Reference [2], except references are made to sections within the single COMMON block rather than to various different COMMON blocks.

Subroutines POWSET, GETIVA, and CLSET are each called once, at the beginning of the program. Subroutine POWSET sets up the array of powers of 2 that is used by the binary search of Subroutine FINDVN. Subroutine GETIVA reads the information file into the arrays IVARQQ and IVARQ. Subroutine CLSET takes information on the lengths of the different sections of the COMMON block and computes working variables that will be used to help identify storage locations for aggregated values.

Subroutine FINDVN is called frequently in the program. Given the name of a NAVMOD input, FINDVN finds the order number of that input in the alphabetical list of NAVMOD inputs (array IVARQQ). This order number is then used to access information stored in the array IVARQ. Subroutine FINDVN returns zero if the symbolic name given it is not a NAVMOD input.

Subroutine PUTONE is called each time the aggregator computes a value for a NAVMOD input scalar or array element (limit variable, resource or ordnance variable, or effectiveness variable). The NAVMOD name and array subscript values are known by the calling routine. Subroutine PUTONE accesses the IVARQ information. From this and the subscript values, it computes the appropriate cell in the COMMON block in which to store the computed value, and stores the value in this cell (see also Chapter II, Section C.3.a).

Table III-12. DEFINITIONS OF IVARQQ AND IVARQ ARRAYS

Storage Location	Contents	Example (JV=11)
IVARQQ(JV)	Alphanumeric name of the JVth variable (input to NAVMOD)	AAPKAD
IVARQ(JV,1)	Number of the section (within the blank COMMON block) that contains the JVth variable)	6
IVARQ(JV,2)	Size of first dimension of the JVth variable	2
IVARQ(JV,3)	Size of second dimension of the JVth variable	2
IVARQ(JV,4)	Size of third dimension of the JVth variable	0
IVARQ(JV,5)	Size of fourth dimension of the JVth variable[a]	0
IVARQ(JV,6)	One less than the location (within the appropriate section of the blank COMMON block) of the first variable value (in the array)	17
IVARQ(JV,7)	First digit gives variable type (1 = integer, 2 = real) Second digit characterizes status with respect to Subroutine TIMET (0 = replacement, 1 = incremental	20
IVARQ(JV,8)	All but the last digit give the location (within the appropriate section of the blank COMMON block) for the last variable value (in the array). Last digit gives number of dimensions[b]	212

[a]NAVMOD does not (currently) contain any four-dimensional arrays, so all entries IVARQ(JV, 5) are zero.

[b]In array IVARQ, scalars are considered to be one-dimensional arrays, the first dimension of which has size 1.

The output of the aggregated values is performed by Subroutine GETALL. Each call to Subroutine GETALL handles all the array elements of a NAVMOD input variable (for scalars, one element). Subroutine GETALL determines the number of output lines necessary and the particular array elements to appear on each line. For each line, Subroutine GETALL calls Subroutine GETLD1 (mnemonic for get one line of data), which performs the actual write. Subroutine GETLD1 retrieves the appropriate (aggregated) values from the COMMON block and writes them and the array subscripting information on the output file, in a format readable by NAVMOD. (For the specifics of this format, see Appendix A of Reference [1].) Subroutine GETLD1 makes use of Subroutine WRITE2 and Function ICEIL; for the specific way these routines are used, see the computer code.

IV. SELECTED DETAILS AND AUXILIARY PROGRAMS

The preceding chapters contain a thorough description of the aggregator preprocessor methodology and computer program. This chapter presents and discusses some assorted issues that the user of the preprocessor should be aware of. These issues are separated into four broad areas, which are discussed in Sections A through D.

First, INGRES has some restrictions on the INGRES users who can access material in an INGRES database. Section A discusses how these restrictions affect the running of the preprocessor and ways to permit more users to execute the program.

NAVMOD imposes some restrictions on certain of its input variables. Section B discusses some of the implications of these restrictions on the more disaggregated data and the database and suggests settings of certain actual resource types and more disaggregated data values that are consistent with the restrictions.

Section C describes the auxiliary computer programs that have been developed for (optional) use with the aggregator. Some of these programs can be helpful when certain information in the database NAVPRE is to be changed. Some of these programs prepare reports about aspects of the data. The programs that generate the shell files of lines for the more disaggregated data, as mentioned in Chapter I, Section C.2 (and in other places in the text) are also documented here.

The aggregator is a preprocessor for the version of NAVMOD documented in References [1] and [2]. However, the aggregator program can still work if NAVMOD is modified in certain ways, provided that the information in the database NAVPRE is updated appropriately. Even if NAVMOD itself does not change, the user might desire to change some of the database information, such as the actual resource types considered or the added indices for effectiveness variables. In such cases, additional updates to the database might be required to maintain consistency (the database is not completely normalized). Changes to the input data files might also be required. Section D discusses the issues involved in maintaining data consistency in some of the more common NAVMOD and database changes. Certain procedures have been developed to address some of these issues; Section D describes these procedures. Some of the procedures use

the auxiliary programs discussed in Section C, and thus Section D refers to Section C, where appropriate. The reader might wish to read Section D before Section C.

A. MULTI-USER ISSUES

1. The Database Administrator and MODIFY Commands

As indicated in Reference [5], Chapter 16, an INGRES table can have any one of a number of different storage structures. Certain storage structures can significantly speed up the retrieval of data from a table. A user with sufficient privilege can change the storage structure of a table, by executing a MODIFY command on it. MODIFY commands can be executed interactively or can be embedded in a program. In the latter case, the user running the program must have sufficient privilege to modify the storage structure.

If all the data are deleted from an INGRES table or data are read into an INGRES table, it is highly desirable to employ certain MODIFY commands, to avoid unreasonably slow program execution. In the aggregator, in the main program and in Subroutine EFFVAR, data are deleted from and/or read into INGRES tables RDATROL and RDATEFF, and the aggregator computer program contains appropriate embedded MODIFY commands. (The other tables in the database have been given storage structures that facilitate execution speed.)

Each INGRES database has one VAX user specified as its database administrator (DBA). Currently, the database administrator for the database NAVPRE can execute MODIFY commands on the tables in NAVPRE, interactively or when running the aggregator program--but other users cannot do this. This limitation may restrict the number of other users who can run the program.[1]

One solution is to have only the database administrator run the program. Note that the DBA is indicated by a particular INGRES user code,[2] which in turn is associated with a

---

[1] Strictly speaking, a MODIFY command on a table can be executed by the owner of a table or by the database administrator acting as though he/she were the owner (see Reference [6], Sections 4.33 and 4.34, and Reference [9], Section 6.7). Currently, the database administrator is the owner of all the tables in the NAVPRE database, but in general, it is possible to have tables in the database that are not owned by the database administrator. In this case, it is not possible for the database administrator to act as both the owner of the table and the database administrator in a single run of INGRES. By careful manipulation of table ownership, it might be possible to develop other procedures (besides the one described in this section) by which non-DBA users can run the aggregator program reasonably quickly. This is an area for further research.

[2] This situation might change in INGRES Release 6.

VAX user name, which conceivably can apply to more than one human being--different people can log in (at different times) under the database administrator's user name.

Having several different people, each using his/her own VAX user name, capable of running the aggregator program is possible in the current version of the program. Section A.2 indicates the procedure required to implement this feature. However, this procedure might be subject to change in future versions of the program, and in future releases of INGRES, and should be regarded as temporary. It is assumed that different users will not be trying to access the database at the same time.

2. Program Procedure For a Non-Database Administrator User

The main idea behind the procedure is to have the more disaggregated data read into the appropriate INGRES tables (RDATROL and/or RDATEFF) by the DBA before program execution starts rather than during program execution. Also, additional options for the input options guide file (Chapter I, Section C.5) have been developed. If these options are used, the program skips the embedded MODIFY commands it might otherwise encounter, thus allowing a non-DBA user to run the program. Of course, the DBA can also run the program using these new options, if desired.

Recall that the aggregator preprocessor has two different parts; the procedures are different for each part, and are explained in Sections 2.a and 2.b. Finally, Section 2.c indicates how a non-DBA user can run the whole program. In all cases, the user should be registered as a valid INGRES user by the INGRES system manager.

a. First Part of Aggregator (Resources, Ordnance, and Limit Variables)

Before the program is run, the DBA should perform each of the following steps. (These steps can be performed with interactive SQL, as documented in References [5] and [6].)

1) Create a permit to the particular user in question for update permission on all of the tables in the database. (See Reference [6], Section 2.7, for information on permits.)
2) Execute the command MODIFY RDATROL TO TRUNCATED, to clear any old information in INGRES table RDATROL.
3) Read (using the "COPY TABLE" command) into INGRES table RDATROL the desired file of more disaggregated data for the resource, ordnance stock, limit, and allocation fraction variables. This file should be prepared as discussed in Chapter II. The correct format for the read can be found in the computer code of the main program or can be constructed as indicated in Reference [6].

4) Execute the command MODIFY RDATROL TO ISAM UNIQUE ON VNAME, COMP1, COMP2, COMP3, COMP4, which will speed up program execution.

Following these steps, an input options guide file should be constructed. This file should consist of a single line, with '10' in columns 1 and 2. The user should have at least read permission (in the VAX command language sense, not in the INGRES sense) on this file. The user should also have at least read permission on the information file, discussed in Chapter I, Section C.4.a. Before the aggregator program is run, the input options guide file should be associated with logical unit 3 and the information file, with logical unit 22, at the command language level.

b. Second Part of Aggregator (Effectiveness Variables)

This section discusses the procedures to follow before the second part of the aggregator program is run. Note that the second part of the aggregator should not be run until after the first part has been run, whether by the DBA or by another user. The preparations discussed in this section, however, are essentially independent of those for the first part, and can be performed before the first part is run. (See also Section 2.c, below.)

First, it should be decided which effectiveness variables should be aggregated. A file containing (at least) the more disaggregated data lines for all of these variables should be prepared, in accordance with the structure and format discussed in Chapter II (Section D). The variables on the file should be consonant with the option specified in the input options guide file. For now, assume that this has been done. The DBA should then perform each of the following steps (using interactive SQL):

1) Create a permit to the particular user in question for (at least) select permission on all of the tables in the database.

2) Execute the command MODIFY RDATEFF TO TRUNCATED, to clear any old information in INGRES table RDATEFF.

3) Read (using the "COPY TABLE" command) into INGRES table RDATEFF the file of more disaggregated data for the desired effectiveness variables. The correct format for the read can be found in the computer code of Subroutine EFFVAR or can be constructed as indicated in Reference [6].

4) Execute the command MODIFY RDATEFF TO ISAM UNIQUE ON VNAME, COMP1, COMP2, COMP3, COMP4. This will speed up program execution--when the program itself is run. Note, however, that this MODIFY command can take some time to execute and might require considerable storage overhead on the INGRES disk, especially if the file of more disaggregated data is large.

When these steps have been completed, an input options guide file should be constructed. The first line of this file should contain an option code number, in columns 1 and 2. Several different options exist, identified by the code numbers 11, 12, 13, and 14. (The reader should review the discussion of the input options guide file in Chapter I, Section C.5, and the discussion of Subroutine EFFVAR in Chapter III, Section G, which explains how the older options on the input options guide file affect the set of variables for which aggregated values will be computed.) New options 11, 12, and 13 operate in the same way as options 1, 2, and 3, respectively, except no files are read into INGRES table RDATEFF during program execution. Instead, the program assumes that the appropriate information has been placed in table RDATEFF before program execution. Thus, if option 12 is invoked, the subsequent lines of the input options guide file should indicate a list of file names. Subroutine EFFVAR will retrieve from INGRES table EVARINFO the variables associated with those file names--and the file read into table RDATEFF before program execution should contain the more disaggregated data values for all of these variables.

If option 13 is invoked, the subsequent lines of the input options guide file should indicate a list of variable names, and the file read into table RDATEFF should contain the more disaggregated data values for all of these variables.

To invoke option 11, the input options guide file should consist of a single line, with '11' in columns 1 and 2. Subroutine EFFVAR will retrieve all of the file names from INGRES table EVARINFO and will retrieve all of the variable names from these files. Assuming that each effectiveness variable has an associated file listed in table EVARINFO, the aggregator program will attempt to compute aggregated values for all of the effectiveness variables, and thus all of the more disaggregated effectiveness data should have been read into table RDATEFF by the database administrator.

If new option 14 is selected, all of the effectiveness variables will be processed, in alphabetical order. To select this option, the input options guide file should consist of a single line, with '14' in columns 1 and 2. All of the more disaggregated effectiveness data should have been read into table RDATEFF by the DBA.

Note that if all of the more disaggregated effectiveness data are read into table RDATEFF, options 12 and 13 can still be run--the extra data may simply increase the program running time.

As was the case with the first part of the aggregator (discussed in Section 2.a), the user should have at least read permission on the input options guide file. The user should also have at least read permission on the information file, discussed in Chapter I, Section C.4.a. Before the aggregator program is run, the input options guide file should be associated with logical unit 3 and the information file, with logical unit 22, at the command language level.

c. Running Both Parts Together

If the input options guide file consists of a concatenation of a guide file for the first part and a guide file for the second part, then the whole aggregator program will be run. The contents of both INGRES tables RDATROL and RDATEFF should be prepared as described in the preceding paragraphs.

B. DEALING WITH RESTRICTIONS ON VARIABLES

NAVMOD imposes some restrictions on certain of its input variables. Violations of these restrictions can cause execution time errors or unreasonable results. This section discusses some of the implications of these restrictions on the more disaggregated data and the database and suggests settings of certain actual resource types and more disaggregated data values that will help produce aggregated data that are consistent with the restrictions.[3] Some of the procedures described in this section recommend changing the value of a variable in the aggregated data. Recall (from Appendix A of Reference [1]) that this can be done by putting an additional data line or lines at the end of the NAVMOD input data file; if more than one value is specified for a NAVMOD input variable or array element, the last value appearing in the file is used.[4]

1. Ordering of Certain Types of Resources

a. Blue Land-based Fighter Aircraft

NAVMOD can model attacks on Red surface ships by Blue land-based aircraft, comprising fighter aircraft and ASW aircraft used in an ASuW role. There are an input

---

[3] The resultant aggregated data set should be checked to ensure that it satisfies the restrictions. Automated data checking procedures for this could probably be developed.

[4] Strictly speaking, this is true for variable values at the beginning of the simulation and for mid-run (Subroutine TIMET) changes to replacement-type variables; see Appendix A of Reference [1]. The aggregator deals only with variable values at the beginning of the simulation.

number (given by the limit variable NKBDPL) of notional types of Blue land-based fighter aircraft. As described in Reference [1], Chapter III, Section A.3.b, when NAVMOD models this combat interaction, it allocates the Blue land-based aircraft between attack and escort missions, in such a manner that fighter aircraft of lower-numbered notional types are more likely to be allocated to escort missions and fighter aircraft of higher-numbered notional types are more likely to be allocated to attack missions.

In the aggregator, the relation of actual and notional types of Blue fighter aircraft, as specified through the allocation fraction variable FPLBLB, should be consistent with this convention of NAVMOD. That is, lower-numbered notional types of fighters should correspond to those actual types that, in fact, would be more likely to fly escort missions, and higher-numbered notional types of fighters should correspond to those actual types that, in fact, would be more likely to fly attack missions. The actual resource types need not be specified in this order (in table GENTYP0), but the notional resource types, as constructed through the allocation fraction variable FPLBLB, should be. (The reader might wish to review Chapter II, Section C.3.c.)

It might be desirable to consider the B-52 (or some other land-based bomber) as one of the Blue land-based aircraft used for ASuW. To model (in NAVMOD) a type of aircraft that does not perform escort or ASW missions, it is recommended that this aircraft type be considered the last notional type of Blue land-based fighter. This can be implemented in the aggregator as follows. First, the B-52 should be listed as an actual resource type in the resource category BLFTR. Next, the desired number of notional (NAVMOD) types of Blue land-based aircraft that are really fighters is determined, and m is used to denote this value plus one (i.e., m encompasses an additional notional type for the bomber). In the file of more disaggregated data for resource, ordnance, limit, and allocation fraction variables, the limit variable NKBDPL is set equal to m, the allocation fraction variable FPLBLB(B-52,m) is set equal to 1, FPLBLB(B-52,j) is set equal to zero for all other notional types j, and FPLBLB(i,m) is set equal to zero for all other actual resource types i. Appropriate values for the B-52 can be specified in the more disaggregated effectiveness data. After the aggregator has been run, the following variables are updated in the aggregated (NAVMOD) data; ELBEE(m) is set equal to zero and PLFDLL(LB,L,m) is set equal to zero for all values of LB and L (from 1 through NLOC) and for the given value of m. These settings will ensure that NAVMOD models only attack missions for the B-52. Section B.3.a discusses another convention that affects the modeling of Blue land-based aircraft on ASuW missions.

b. Red Bombers

NAVMOD can model several different types of Red bombers. The number of types modeled is given by the input limit variable NKRB. As discussed in Reference [1], Chapter III, Section A.8.f, NAVMOD assumes that lower-numbered notional types of Red bombers are more effective (in terms of speed and launch range) than higher-numbered notional types. As far as NAVMOD is concerned, these assumptions are met if the values of certain NAVMOD input variables that concern the speed and range of bombers satisfy certain relationships. These relationships, and the implications for the more disaggregated data, are discussed in Section B.2.d.

2. Restriction Patterns for Variables--Implications for the More Disaggregated Data

a. Variables That Must be Between Zero and One

Many of the NAVMOD inputs represent probabilities, fractions, or proportions and should have values between zero and one (inclusive). (Appendix F of Reference [2] indicates these variables.) The aggregation process is merely a weighted averaging. Therefore, if all of the more disaggregated data values for a variable are between zero and one (inclusive), then the aggregated value will be also.

The variables CVREPP and URGEP have a special form of restriction. In NAVMOD, these are both 6-vectors. For each variable, the first three elements must be between zero and one (inclusive), but there are no restrictions (except nonnegativity) for the second three elements. In the more disaggregated data, the user might wish to put added indices on these variables. The first index, however, is a numerical index. If every more disaggregated data line in which the first component name is '01', '02', or '03' is given a data value between zero and one, the resultant aggregated values will satisfy the restrictions.

b. Variables That Must be Strictly Positive

Some of the NAVMOD inputs--at least certain array elements of these inputs--must have values strictly greater than zero. Appendix F of Reference [2] indicates if a variable has this restriction, and Table B-9 of Reference [2] lists the variables that do. (As that table indicates, some of the variables are used only in certain combat options and need have strictly positive values only if actually accessed by the NAVMOD code.) If at least some

data lines appear in the file for the variable, and all of the more disaggregated data values for the variable (on the lines in the file) are strictly positive, and at least some nonzero weight goes to these values, then the aggregated--weighted average--value will also be strictly positive.

c. Integer Option Indicator Effectiveness Variables

A number of the integer inputs to NAVMOD can be given values only from a small set of allowable values. (Many of these inputs govern the selection of combat options.) The allowable values appear in the definitions of the variables.

As stated in Chapter II, Section C.3.a, if these integer variables are not given any added indices, the values in the more disaggregated data will simply be copied to the aggregated data. It is not anticipated that added indices would be used for these variables, thus the user need merely check that the value on each more disaggregated data line is from the appropriate allowable set.

d. Variables With Extensive Restrictions

Certain NAVMOD inputs have extensive restrictions associated with them. These variables are listed in Table B-10 of Reference [2], which for convenience is reprinted as Table IV-1 here. It is probably best to check and adjust the aggregated data for these variables, after the aggregator has been run, rather than spending too much time adjusting the more disaggregated data. (As mentioned earlier, automated data checking procedures have not yet been developed, but it would probably be easy to do so.)

The variables D1T, D2T, D3T, VBT, and VMT in Table IV-1 deal with the effectiveness of Red bombers, as mentioned in Section 1.b. Each notional type (KRB) of Red bomber corresponds to a mix of actual types of bombers, as determined by the allocation fraction variable FATABT (of course, a mix can in fact consist of only one actual type of bomber.) The user can adjust the elements of FATABT to let the more effective actual types of bombers correspond to lower-numbered notional types, while setting the more disaggregated data for the above variables consistent with the actual types. This adjustment will increase the chance that the resultant aggregated data for these variables will satisfy the restrictions in Table IV-1. However, the user should still check these aggregated data.

Table IV-1. INPUT VARIABLES WITH EXTENSIVE RESTRICTIONS

Input Variable(s)	Used in Routine	Restrictions and Comments
DDFRAB(KRS,KBS)	DDAY	For each value of KRS from 1 through NKRS the sum $$\sum_{KBS=1,NKBS} DDFRAB(KRS,KBS)$$ must not exceed 1 (NKBS=NKBES+3). (This insures that the number of Red ships attacking is not greater than the number of Red ships present, for each type of Red ship.)
D1T(I,KRB) D2T(I,KRB,ICV) D3T(KRB) VBT(KRB) VMT(KRB)	CTFMOD	If NKRB $\geq$ 2, then for KRB=1,...,NKRB−1, the relationships: D1T(I,KRB) $\geq$ D1T(I,KRB+1), for I=1,2, D2T(I,KRB,ICV) $\geq$ D2T(I,KRB+1,ICV) for I=1,2 and all ICV, [D1T(1,KRB)−D3T(KRB)] $\geq$ [D1T(1,KRB+1)−D3T(KRB+1)], VBT(KRB) $\geq$ VBT(KRB+1), and VMT(KRB) $\geq$ VMT(KRB+1) must hold. (This insures that different types of bombers are ranked in decreasing order of particular types of effectiveness.) For KRB from 1 through NKRB, D1T(I,KRB) (for I=1,2) and VBT(KRB) must strictly exceed zero. VMT(KRB) must strictly exceed zero if ITFFKM $\geq$ 1, and D3T(KRB) must be less than or equal to D1T(1,KRB). Also, D1T, D2T, D3T, VBT, and VMT are physical quantities and input values should be compatible with physical units.
ELFBL0(LB,LA,KBA)	ELSLBA	For each region LB=1,...,NLOC, and each type of Blue land-based aircraft KBA=1,...,NKBDPL+1, the sum $$\sum_{LA=1,NLOC} ELFBL0(LB,LA,KBA)$$ must not exceed 1. (This insures that no more aircraft are assigned to combat than are available.)
NPPFFX PPCVFX(I)	POWERP	Let NPPFU be defined as min{NPPFFX,MXTABP} if NPPFFX $\geq$ 1 and as MXTABP if NPPFFX=0, where MXTABP is a strictly positive integer symbolic constant. Then components 1 through NPPFU of the vector PPCVFX must form a monotone increasing sequence, and if NPPFU $\geq$ 2, the differences PPCVFX(I+1)−PPCVFX(I) must form a monotone decreasing sequence, for I=1,...,NPPFU−1. (This guarantees that the value of ground targets destroyed is a concave increasing function of power projection sorties.)
SSFEAK(KBA,KRSS) SSFUAK(KRSS) SSSORR(KBA)	SHPSHP	If ISAAM=1 and NSAG=0, then for each value of KRSS from 1 through NKRSS the relationships SSFUAK(KRSS) $\leq$ min{1,1/SSSORR(1)}, and SSFEAK(KBA,KRSS) $\leq$ min{1,1/SSSORR(KBA)}, for KBA=1,2 must hold.

e. Interrelationships Among Different Variables

In addition to listing restrictions on individual variables, Table IV-1 has listed some restrictions on relationships between certain variables. For maximum consistency of NAVMOD results, there are a number of other groups of NAVMOD input variables whose values should satisfy certain relationships. The text of Reference [2] discusses many of these groups.

For example, consider the two variables, SBORCM, the minimum number of SLCMs that a Red missile-firing submarine must carry in order to attack the task force, and ZMPSTG, the number of missiles that a Red missile-firing submarine fires at the task force (if it makes an attack). As discussed in Reference [2], Chapter II, Section C.6.a, SBORCM should be greater than or equal to ZMPSTG, and the two variables could reasonably have the same value.

In the more disaggregated data, the user might want to put added indices (e.g., for type of Red missile-firing submarine) on both of these variables. If the same added indices are chosen for each variable, and the corresponding more disaggregated data elements have values that satisfy the desired restriction, chances are that the aggregated data will also satisfy the restriction. This is not guaranteed, however. If unreasonable NAVMOD results occur, the user should check the aggregated data (i.e., the NAVMOD input data) and check that the appropriate data interrelationships, as indicated in Reference [2], hold. The index in Chapter IV of Reference [2] might be helpful in indicating the places in the text where variables are discussed.

f. Limit Variables

Chapter II (especially Section C.3.b) and Chapter III (Sections C and D) have discussed limit variables in detail. This section makes a few additional remarks about some of them, focusing on their use in NAVMOD.

NAVMOD requires certain of its limit variables to have values of 1 or greater. Also, a limit variable value of zero might result in the aggregator's not aggregating certain data. Accordingly, Subroutine LMVSET of the aggregator puts a floor of 1 on all limit variables: if a value of zero is specified in the input file, Subroutine LMVSET will give the limit variable a value of 1. (It would be easy to change this feature of Subroutine LMVSET if desired.)

However, NAVMOD does allow certain limit variables to be zero. If this becomes desirable, the aggregated data can be updated. Because of the indexing of certain variables, however, some caveats apply.

The following limit variables can simply be reset to zero in the aggregated data, if desired:

- NSAG, the number of SAGs (should be zero if it is not desired to model Red surface ships as being organized in SAGs);
- NTC, the number of types of SLOC-delivered cargo (can be zero if NAVMOD SLOC model is not being exercised);
- MIMPSG(ISAG), which corresponds to NMPSG in the aggregator, is the number of movement periods for a SAG--which is irrelevant if NSAG is zero.

The limit variable NKBES is the number of notional types of Blue escort ships that NAVMOD models. As discussed in Chapter III, Section A, of Reference [2], NAVMOD can accept a zero value for this variable. The value produced and used by the aggregator will not be zero, however. Because of the indexing convention on many variables involving Blue ships, the value of NKBES should not be changed in the aggregated data. If it is desired not to simulate escort ships in a NAVMOD run, the user can reset (all elements of) the NAVMOD input resource variable BESS, the number of escort ships, to zero in the aggregated data file.

Similarly, the number of notional types of Red ships simulated, NKRS, will be computed by the aggregator (from the aggregator input NKRSN, as discussed in Chapter II, Section C.3.b) to encompass two types of Red submarines, one type of Red resupply ship (which will be the NKRS-th notional type of Red ship) and at least one type of Red surface ship that is not a resupply ship. (The number of types of such surface ships is given by max{NKRSN,1}.) The value of NKRS should not be changed in the aggregated data. If it is desired not to simulate a certain type of ship, the user can set to zero the appropriate components of the NAVMOD resource variable RS(KRS,L), the number of Red type-KRS ships in Region L. Set RS(NKRS,L) to zero for all L if it is desired not to have Red resupply ships (but in this regard, the reader might wish to review Chapter II, Section B, of Reference [2]).

NAVMOD has certain special conventions for the six limit variables NPPFFX, NTABA, NTABD, NTABH, NTABS, and NTABV. The relevance of these conventions to the aggregator is discussed in the next section.

g. Specially Indexed NAVMOD Inputs

The six NAVMOD variables PDINV, PPCVFX, TAB10T, TAB12, TAB13T, and VTTAB have certain indexing patterns in common. This section should be read in conjunction with Chapter III, Section C.4, of Reference [2], which gives information on how NAVMOD uses these variables. As discussed there, associated with each of these variables is: 1) a symbolic constant, whose value is used as a dimension size for the variable and 2) a limit variable, which gives the number of components actually used by NAVMOD. (Let us call PDINV, PPCVFX, etc., the base variables here, to distinguish them from the limit variables.) The limit variables are NPPFFX, NTABA, NTABD, NTABH, NTABS, and NTABV, and the associated symbolic constants are MXTABP, MXTABA, MXTABD, MXTABH, MXTABS, and MXTABV.

In the aggregator, each of the symbolic constants is interpreted as a numerical (code 12) index. Each of these indices is used in exactly one variable. Each limit variable is associated (in table LIMVINFO) with the corresponding index. Values for these limit variables should be entered in the file of more disaggregated data for the resource, ordnance, limit, and allocation fraction variables.

NAVMOD allows zero values for the limit variables but interprets a value of zero as a signal to use the corresponding symbolic constant value. This convention cannot be exploited in the aggregator. The limit variable values input to the aggregator should not be zero but should equal the number of components of the associated variable that are desired.

Added indices can be specified for the base variables as desired. The use of added indices here can allow the more disaggregated data for PPCVFX, PDINV, TAB12, and VTTAB to depend on a relevant actual resource type; this approach might be appropriate. TAB10T and TAB13T are two-dimensional arrays in NAVMOD, and the second dimension of each variable corresponds to a resource-based index in the aggregator, but it still might be appropriate to specify some added indices for these variables.

As discussed in Chapter III, Section C.4, of Reference [2], the base variables (i.e., the aggregated, NAVMOD input data for these variables) must satisfy certain restrictions. (The restrictions on PPCVFX are also shown in Table IV-1.) After the aggregator has been run, the user should check that the aggregated data satisfy these restrictions. If the more disaggregated data satisfy these restrictions for each possible combination of

component names for the added indices, the aggregated data should satisfy the restrictions--but the user should still verify that the aggregated data satisfy the restrictions.

The determination of the more disaggregated data values for the variables TAB10T and TAB13T can be simplified by exploiting some properties of the treatment of these inputs by the NAVMOD code. Recall the definitions of these variables as they are used by NAVMOD itself.

> TAB10T(I,K) is the number of antiship missiles that can be engaged by I AAW escorts given that these missiles were launched from type-K Red bombers, for K=1,...,NKRB. For K=NKRB+1, TAB10T is this number given that the missiles were launched from Red submarines.
> If TAB10T(2,K) < TAB10T(1,K) then the code acts as if TAB10T(I,K) = I*TAB10T(1,K) for all I.

> TAB13T(I,K) is one minus the relative reduced carrier effectiveness if a carrier receives I hits by antiship missiles launched from type-K Red bombers (in Subroutine CTFMOD), for K=1,...,NKRB. TAB13T(I,NKRB+1) is for missiles from Red submarines. Each component should be between 0 and 1, inclusive. If TAB13T(2,1)=1 but TAB13T(1,1) < 1, then only the values of TAB13T(1,K) (for each platform type K) are used, and the effects of different shots are assumed to be independent of one another. (Otherwise, for maximum consistency of results, components (I,K) should form a nonincreasing sequence in I for each value of K.)

If the special conditions indicated in the preceding paragraphs are fulfilled, only the values for component I=1 will be used by NAVMOD--and thus more disaggregated data (input values for the aggregator) need be procured only for components I=1. However, some editing of the aggregated data file might be necessary. The user should proceed as follows.

To implement the special condition for the variable TAB10T, set the value of NTABA (in the file of more disaggregated data for resource, ordnance, and limit variables) equal to one. In the more disaggregated effectiveness data for TAB10T, find the data lines where the first component name is '01' and set the data values in these lines to appropriate values (reflecting the type of Red platform and the actual resource types associated with any added indices). The aggregator will ignore data lines for TAB10T with the first component

name '02' or greater. After the aggregator is run, no editing of the file of aggregated values is necessary (NAVMOD will ignore the value of NTABA if the TAB10T values satisfy the special condition).

To implement the special condition for the variable TAB13T, set the value of NTABH (in the file of more disaggregated data for resource, ordnance, and limit variables) equal to one. In the more disaggregated effectiveness data for TAB13T, find the data lines where the first component name is '01' and set the data values in these lines to appropriate values (reflecting the type of Red platform and the actual resource types associated with any added indices). Be sure, however, that these data values are strictly less than one.[5] The aggregator will ignore data lines for TAB13T with the first component name '02' or greater. After the aggregator has been run, edit the file of aggregated values, and set the value of TAB13T(2,1) equal to 1. (NAVMOD will ignore the value of NTABH if the TAB13T values satisfy the special condition.)

3. On the Aggregator's Treatment of Certain NAVMOD Variables

A few NAVMOD input variables (or array elements of certain variables) are used in such a way by NAVMOD that the aggregator methodology is not ideally suited to developing data values for them. The aggregator will accept more disaggregated data for these variables or array elements and will compute and output aggregated values for them, but these values might not be completely appropriate. Values for these variables can be developed by other means, and edited into the aggregated data file.

This section briefly discusses these variables. Definitions of all variables appear in Appendix F of Reference [2]. Adapting the aggregator methodology to provide a more automated way of producing data values for these variables may be possible at some point in the future.

a. Variables With the Index MZKBD2 (Blue Land-based Aircraft)

One of the combat interactions that NAVMOD models is Blue land-based aircraft attacking Red surface ships. The Blue aircraft that can be involved are the land-based fighters plus the land-based ASW aircraft, acting in an ASuW role. Combat can occur in every region--the region where the task force is currently located and other regions.

---

[5] Recall from Table II-27 that the adaptive index JTASCM has been suggested as an added index for TAB13T. Use of this added index would not interfere with the current procedure.

Certain variables used in the modeling of this interaction have a dimension index MZKBD2. In NAVMOD, this index is interpreted as follows. Elements 1 through (the input limit variable) NKBDPL correspond (respectively) to the NKBDPL types of Blue land-based fighter aircraft, element NKBDPL+1 corresponds to Blue ASW aircraft operating in regions other than the region where the task force is currently located, and element NKBDPL+2 corresponds to Blue ASW aircraft operating in the region where the task force is currently located. (The rationale is that ASW aircraft in the latter category might be configured primarily for ASW support to the task force, rather than ASuW, and thus might have ASuW effectiveness different from that of ASW aircraft performing ASuW in other regions.) The variables are EAAKAD, EAAMPA, EAOBAA, EAOBAF, ELAPMU, ELASPS, ELBEA, and ELLFAV. Definitions of them appear in Appendix F of Reference [2].

In NAVMOD, each of the preceding variables is a one-dimensional array, with the index MZKBD2. Added indices can be specified as desired. The way the aggregator is currently programmed, however, the aggregated value computed for element NKBDPL+2 of each of these variables will be the same as the value computed for element NKBDPL+1.[6] This is true regardless of which, if any, added indices are specified. Some amount of reprogramming of the aggregator, as well as changes to the database, would be necessary to change this convention. If separate values for the NKBDPL+2 elements are desired, they should be developed via side calculations and edited into the NAVMOD input file.

b. Variables for Task Force Aircraft Capacities

The four variables AEWACI, ASWACI, ATTCKI, and FGHTRI relate to the capacities of the (Blue) task force for various types of sea-based aircraft. (In NAVMOD, all of these variables are scalars.) The variable AEWACI is defined as the total number of AEW aircraft that would be carried on all of the carriers in the task force if all carriers were undamaged and all were carrying a full load of aircraft. The definitions of ASWACI, ATTCKI, and FGHTRI are similar, for ASW aircraft, attack aircraft, and fighter aircraft, respectively. For more information on how NAVMOD uses these variables, see Chapter II, Section B.3, of Reference [1].

---

[6] As one might expect, the more disaggregated data for these variables do not distinguish region of the task force as opposed to other regions.

These variables have important implications for the numbers of aircraft that NAVMOD models the carriers as launching, and they should not have zero values. Unfortunately, the aggregator methodology is not well suited for computing values for them. It is evident from the definitions that the values for these variables really depend on the number of carriers in the task force and represent totals over all of the carriers present. Available data might well be per carrier and would vary with the particular carrier and actual type of aircraft. The weighted averaging procedure of the aggregator is not well suited to computing a data value that represents a sum. It is therefore recommended that data for the these four variables be developed separately. (It might be possible to develop another auxiliary program that would accept inputs for actual carriers and aircraft types, and add them to produce values suitable for NAVMOD. Or, the aggregator program could be changed to deal with these four variables explicitly.)

c. Treatment of Selected Other Variables and Indices

Chapter I discusses the fact that the structure of the aggregator arose out of the presence of certain broad patterns of regularity in the dimensioning of the NAVMOD input (array) variables. A few NAVMOD input variables do not fit easily into these patterns. This section briefly discusses some of these variables. As with all of the variables discussed in this section (Section B.3), data values can be developed independently of the aggregator and can be edited into the file of aggregated data.

(1) Variables TRSBA0 and ARSBA

Recall that NAVMOD does not model Red sea-based aircraft at the same level of detail as some of the other resources. The reader might wish to review Chapter III, Section B.2, of Reference [1], which discusses NAVMOD's modeling of Red sea-based aircraft. In future versions of NAVMOD and the aggregator, Red sea-based aircraft may possibly be treated in the same manner as the other resources, but this is not currently the case. NAVMOD does not have an explicit input resource variable that represents a stock of Red aircraft; accordingly, the aggregator does not have a resource category for Red aircraft. (This is, in some sense, a limitation of NAVMOD and the aggregator, and a correctable one, but the discussion here applies to the current versions of these programs.)

As discussed in Chapter III, Section B.2, of Reference [1], however, two variables, TRSBA0 and ARSBA, are used to indicate numbers of Red sea-based aircraft. The initial number of Red sea-based aircraft is set to the value of TRSBA0 (mnemonic for

total Red sea-based aircraft initial) if this value is nonzero; otherwise, this initial number is computed as the total number of spots for aircraft on the Red ships. The variable ARSBA (mnemonic for added Red sea-based aircraft) can be used to specify an increment to the number of Red sea-based aircraft at a non-initial time period.

In NAVMOD, the variables TRSBA0 and ARSBA are both scalars. In the aggregator, ARSBA is irrelevant (see the following paragraphs), and it is recommended that TRSBA0 remain a scalar, without added indices. A reason for this is as follows. A detail (code 30) index (named JARSBA) or actual type of Red sea-based aircraft has been set up in the aggregator. This index can be used as an added index for certain effectiveness variables, as appropriate. (The user can edit the INGRES table INDXDTYP to reflect the desired actual types of Red sea-based aircraft to be modeled, and their relative weights.) However, TRSBA0 represents a total number of average aircraft. If the index JARSBA were used on TRSBA0, the aggregator would average the different values input for the different actual types of Red aircraft; it would not sum them.

It is therefore recommended that data for TRSBA0 be developed independently of the aggregator and that no added indices be specified for TRSBA0. The user could edit the file of aggregated data directly. Alternatively, if these recommendations are followed, the more disaggregated data for TRSBA0 will consist of merely one data line. The appropriate data value can be edited into this line, and the aggregator will copy it to the file of aggregated data. It is not necessary for the value of TRSBA0 to be nonzero in order for NAVMOD to model Red sea-based aircraft. (The user should be aware of the input variable RAPRS, however, and might wish to add the index JARSBA to this variable.)

The variable ARSBA is used only at non-initial time periods in a NAVMOD simulation. The aggregator outputs only initial values. Thus any value the aggregator computes for ARSBA will be ignored by NAVMOD itself.

(2) Variable IPRSRA

The integer variable IPRSRA (mnemonic for indicator for priority for sheltering Red aircraft) indicates a priority ordering for sheltering the various (notional) types of Red (land-based) aircraft. It is used in NAVMOD's airbase attack routine, as described in Chapter III, Section A.4.c, of Reference [1].

In NAVMOD, this is a vector (one-dimensional array). The number of elements used by the NAVMOD code is NKRB+2, which comprises NKRB notional types of

bombers, one notional type of fighter, and one notional type of interceptor. (NKRB is an input limit variable to both NAVMOD and the aggregator.) The appropriate values of the elements of IPRSRA depend on the types of notional Red aircraft simulated--on the mixes of actual aircraft types that make up the notional types. To develop values for IPRSRA, one should first know what these mixes are. This is possible, via the allocation fraction variable FATABT and the more disaggregated data for the resource variables ATABT, AESCAB, and AINTCP.

In the aggregator, IPRSRA has been set up as a one-dimensional array, with the index MZIPR. This index has been declared as code 16 (compound numerical) in table INDXBINFO. Subroutine LMVCMP sets its value equal to NKRB+2, and Subroutine AGGEFF will simply copy any values specified in the more disaggregated data files, for elements 1 through NKRB+2.

(3) Variable SLCCS--NAVMOD SLOC Model

As described in Chapter III, Section B.3, of Reference [1], NAVMOD has as an option a simple model of SLOC operations. If this option is exercised, the input variable SLCCS(KBS,ITC) (mnemonic for SLOC-delivered cargo size) is used to represent the number of units of SLOC-delivered cargo of type ITC that one type-KBS Blue ship carries. Here ITC runs from 1 through (the value of) NTC, an input limit variable that represents the number of types of SLOC-delivered cargo modeled. The variable SLCCS is dimensioned as SLCCS(MZKBS,MXTC), where the symbolic constant MXTC is an upper limit on NTC. MXTC is listed as an index in the aggregator.

Because SLCCS is the only NAVMOD input with MXTC as an index, it was decided not to create a whole new resource category for Blue SLOC-delivered cargo. Instead, MXTC is declared as a code-12 (numerical) index in table INDXBINFO, with the associated limit variable NTC (as used in NAVMOD). The more disaggregated data for SLCCS have a component name for actual type of Blue ship and a (character-encoded) numerical value for type of cargo. Of course, if one does not want to exercise the NAVMOD SLOC model, no disaggregated data for SLCCS need be input.

(4) Variable PIWACM

The variable PIWACM (mnemonic for power-projection-intercept weight for aircraft and cruise missiles) is used in NAVMOD's power projection routine, as described in Chapter III, Section A.10.b, of Reference [1]. It gives relative weighting factors for

different types of Blue sea-based power projection platforms and is used to allocate Red interceptor sorties between defense against Blue aircraft and defense against Blue cruise missiles. In NAVMOD, the variable PIWACM is a 3-vector. The first element is a relative weight for Blue attack aircraft, the second element is a relative weight for Blue fighter aircraft, and the third element is a relative weight for Blue cruise missiles, encompassing both surface-launched and submarine-launched cruise missiles.

In the DCOMMN.DAT file, PIWACM is dimensioned as PIWACM(MZKBA1). The symbolic constant MZKBA1 has been defined as MYKBA + 1. Since the value of the symbolic constant MYKBA is fixed at 2, the value of MZKBA1 is fixed at 3, which is consistent with the use of PIWACM in NAVMOD. PIWACM is the only variable that uses the symbolic constant MZKBA1.

In the aggregator database, one might set up an index MZKBA1 as a compound resource-based index, that would encompass four resource categories--attack aircraft, fighter aircraft, surface-launched cruise missiles, and submarine-launched cruise missiles. But the variable that uses MZKBA1, PIWACM, only has three notional components, with two different resource categories being considered together in one of the components. This structure is so inconsistent with the indexing patterns used for most variables that a special aggregation algorithm for it would be necessary; however this has not been done. Instead, in table INDXBINFO, MZKBA1 has simply been declared as a code-14 numerical index, with 3 as the (fixed) number of types. In table RVARINFO, PIWACM is listed with the one index MZKBA1. If PIWACM is not given any added indices, then the more disaggregated data for PIWACM will consist merely of three data lines. Appropriate values can be specified in these lines, and the aggregator will copy them to the file of aggregated data.

In view of the fact that PIWACM is the only variable that uses MZKBA1 as an index, this convention seemed reasonable.

(5) Variable SAPBSA

The variable SAPBSA (mnemonic for SAGs per Blue sea-based aircraft) is used in Subroutine SHPSHP of NAVMOD. It is used only if Red surface ships are to be modeled

as being organized in SAGS--only if the input limit variable NSAG is one or greater in a NAVMOD input data set.[7]

This variable represents the average number of Red SAGs that one Blue sea-based aircraft can attack. The reader might wish to review the discussion in Chapter III, Section A.9.b, of Reference [1], on the use of the variable SAPBSA in NAVMOD. As discussed there, the variable is a 3-vector. The first element applies to Blue attack aircraft on attack missions, the second element applies to Blue fighter aircraft on attack missions, and the third element applies to Blue fighter aircraft on escort missions.

This indexing pattern is atypical for NAVMOD inputs, and it does not fit easily into the resource category structure. In the DCOMMN.DAT file, the variable SAPBSA has simply been declared as SAPBSA(3), without a symbolic constant name. Accordingly, in the database NAVPRE (table RVARINFO), SAPBSA has been listed simply with the numerical (code 14) index MY3. If SAPBSA is not given any added indices, then the more disaggregated data for SAPBSA will consist merely of three data lines. Appropriate values can be specified in these lines, and the aggregator will copy them to the file of aggregated data. That is, the aggregator will make no attempt to aggregate across different aircraft-type/mission-type combinations.

C. DESCRIPTIONS OF AUXILIARY PROGRAMS

This section describes the nine auxiliary programs that have been developed to facilitate certain changes to the database NAVPRE. These programs seemed particularly helpful and could be developed quickly. Developing more auxiliary programs may be possible.

The programs are named CHGINP3, CHGEFFDAT, GENGENUST, GENIXRTYP, GENMXVAL, GENEFFDAT, GENROLDAT, GENREPORT, and DIMREPORT. Table IV-2 briefly describes the purposes of these programs. The source code for these programs can be delivered along with the aggregator program itself. Note that CHGINP3 is a FORTRAN program and should be compiled and linked. The other programs are written in FORTRAN with Embedded SQL commands, and should be run

[7] Since NSAG is a limit variable, Subroutine LMVSET will compute a nonzero value for it, as discussed in Section B.2.f. But the user should reset NSAG to zero in the aggregated (NAVMOD input) data if the user does not wish to model SAGs.

Table IV-2. PURPOSES OF AUXILIARY PROGRAMS

CHGINP3--in accordance with changes to the NAVMOD inputs, as specified in the DCOMMN.DAT file, prepares a new information file (BLKINP.DAT) and new included files for NAVMOD, the aggregator preprocessor, and the interaction selector preprocessor.

CHGEFFDAT--prepares a shell file of more disaggregated data lines (with zeros in the space for the data value), for each effectiveness variable on an input list.

GENEFFDAT--prepares shell files of more disaggregated data lines (with zeros in the space for the data value), for all the effectiveness variables, organized by category.

GENROLDAT--prepares a shell file of more disaggregated data lines (with zeros in the space for the data value), for the resource, ordnance, limit, and allocation fraction variables.

GENGENUST--from the information on actual resource types in INGRES table GENTYP0 prepares a file with the appropriate information for (the first four columns of) INGRES table GENUSTYWGT (useful when actual resource types, as specified in table GENTYP0, change).

GENIXRTYP--from the information on actual resource types in INGRES table GENTYP0, prepares a file with the appropriate information for (the first three columns of) INGRES table INDXRTYP (useful when actual resource types, as specified in table GENTYP0, change).

GENMXVAL--reads the values of symbolic constants from the file DCOMMN.DAT, and updates the appropriate information in the INGRES database accordingly (useful if values of symbolic constants change).

GENREPORT--prepares a report of the resource categories and the actual resource types in them.

DIMREPORT--prepares a report showing definitions of NAVMOD input variables (as used in NAVMOD), and definitions of their NAVMOD and added indices.

through the Embedded SQL/FORTRAN preprocessor, the FORTRAN compiler, and the linker, in accordance with the procedure described in Chapter 4 of Reference [10]. When an instruction is made to run an auxiliary program, it is assumed that this has been done. All input files read by the programs are assumed to reside in the appropriate directories.

Note that programs GENREPORT and DIMREPORT merely generate reports on the database and are not used in the procedures to change the data. Program CHGINP3 can also be used to facilitate certain changes to the interaction selector preprocessor and to NAVMOD itself.

The program descriptions here are fairly brief; for more information, the reader is referred to the source code of the programs. The descriptions of the INGRES tables and their columns, given in Chapters II and III and in Appendix B, might also be helpful. The programs can be run by a user other than the database administrator, but such a user should have select and/or update permission on the appropriate INGRES tables, and the appropriate VAX/VMS privileges on the input and output files.

1. Program CHGINP3 ("change inputs--version 3")

Program CHGINP3 is essentially an updated version of the CHGINP program discussed in Appendix C of Reference [2], a review of which might be helpful. (CHGINP2 was an interim modification of the CHGINP program.) Program CHGINP3 reads the file, named DCOMMN.DAT, that contains certain information on NAVMOD inputs, and generates a number of output files: all of the ones CHGINP produces, plus several additional ones that are relevant to the aggregator preprocessor and the interaction selector preprocessor (to which CHGINP3 is also an auxiliary program).

Part of the source code of the aggregator preprocessor is on included files--separate files that are brought into the aggregator code at compilation time by means of INCLUDE statements in the main source code file. Table A-5 of Appendix A presents additional information on these files. Their contents relate to the storage of the NAVMOD variables and depend on the contents of DCOMMN.DAT. If the DCOMMN.DAT file is changed in accordance with the procedures described in Appendix C of Reference [2], and the program CHGINP3 is run on it, not only will all of the necessary files for NAVMOD be generated (as with the CHGINP program), but new versions of the included files for the aggregator computer code (and the interaction selector computer code) will also be generated. (The aggregator program should then be recompiled.)

2. Program GENMXVAL ("generate maximum values")

Many of the symbolic constants in NAVMOD are indices in the aggregator, and the values of these symbolic constants are stored in the aggregator INGRES database, NAVPRE. As discussed in Appendix C of Reference [2], the user can change the values of certain NAVMOD symbolic constants. When these values are changed, the information in NAVPRE that concerns symbolic constants should be updated appropriately, before the aggregator is run again. Program GENMXVAL performs this update. Table IV-3 lists some features of Program GENMXVAL.

The reader is assumed to be familiar with the structure of the file DCOMMN.DAT, as discussed in Appendix C of Reference [2]. Program GENMXVAL reads this file, except for the list of incremental variables. It begins by reading the names of the NAVMOD COMMON blocks that contain input variables, and stores them in a working array. It then reads each symbolic constant name and formula; using Subroutine DECODE it computes from the formula a numerical value for the symbolic constant.[8] The symbolic constant names and numerical values are stored in arrays.

With one exception, numerical values need not be computed for aggregator indices that are not NAVMOD symbolic constants. The exception is the index MXKRSN, which indicates Red non-resupply surface ships. A special section of code sets the numerical value of MXKRSN as one less than the value for MXKRSS (the index for Red surface ships).

Conversely, some NAVMOD symbolic constants do not correspond to aggregator indices. Program GENMXVAL computes numerical values for such constants, but these values are not used to update information in NAVPRE.

When all of the numerical values have been computed, the program takes each symbolic constant name (including MXKRSN), in turn, and sets to the associated numerical value the relevant pieces of information in the database, for the index with the same name as the symbolic constant (if any). For the specifics, see the computer code. Subroutine LMVCMP, which is similar but not identical to the aggregator subroutine with this name, aids in checking the consistency of numerical values for indices with codes 6 or 16.

---

[8] Subroutine DECODE of Program GENMXVAL is identical to Subroutine DECODE of Programs CHGINP3 and CHGINP, and has the same purpose.

Table IV-3. SELECTED INFORMATION ABOUT AUXILIARY PROGRAM GENMXVAL

Purpose: Reads the values of symbolic constants from the file DCOMMN.DAT and updates the appropriate information in the INGRES database accordingly (useful if values of symbolic constants change).

ASCII files read:	DCOMMN.DAT
INGRES tables accessed for retrieval:	RVARINFO INDXBINFO INDXGENUS LIMVINFO CMPLIM
INGRES tables updated:	LIMVINFO (column dimval) INDXBINFO (column mxvalntl; columns ivalnt and nsteps, for some variables) CMPLIM (column stepixval)
ASCII files output:	Logical unit 6--informative messages
Subroutines:	GETNAM DECODE LMVCMP

These database updates are the main purpose of Program GENMXVAL. However, the program also checks that for each NAVMOD input, the sequence of (the first NODN) indices in INGRES table RVARINFO and the sequence of dimension names in the DCOMMN.DAT file match. Subroutine GETNAM aids in this process. If there is a mismatch, a message is printed.

Note that there will be such a mismatch for NAVMOD inputs with dimension sizes that are indicated in DCOMMN.DAT with numbers, rather than with symbolic constants (as discussed in Chapter II, Section B.2). In this case, the number in DCOMMN.DAT will equal the value in column mxvalntl of table INDXBINFO for the corresponding index (from table RVARINFO). (If these two values are not equal, an error has been made in preparing the database, and the database contents should be thoroughly rechecked.)

3. Program GENGENUST ("generate table GENUSTYWGT")

As indicated in Chapter II, Section A, the user can specify which actual resource types are to be considered in each of the resource categories. If changes in these types are desired, the desired new types should be entered in INGRES table GENTYP0, by means of QBF or SQL (References [12] and [5]). Then, auxiliary program GENGENUST

should be run, and its output file should be read into table GENUSTYWGT. (Program GENIXRTYP should also be run, as discussed in Section 4.) Program GENGENUST reads the information on actual resource types in INGRES table GENTYP0. From this, it updates those INGRES tables that hold information on the numbers of actual resource types, and it prepares a file with the appropriate information for (the first four columns of) INGRES table GENUSTYWGT. Table IV-4 lists some features of Program GENGENUST.

Program GENGENUST operates as follows. First, it selects all of the resource category names from table GENUSINFO and stores them in a vector. For each resource category, it retrieves the actual resource types, in order, from table GENTYP0, and stores them. The appropriate number of notional types is determined by a retrieval from table LIMVINFO (column dimval, as the value in column lvval is not known until the aggregator program itself is run). For each notional-type/actual-type combination, the resource category name, the notional type number, the actual type order number, and the actual type name are written on the output file GENUSTYWG.IN. Note that this output file will be located in the device and directory specified in INGRES table PFNAMEF.

After Program GENGENUST has been run, the database administrator should change the contents of INGRES table GENUSTYWGT. This can be done by executing the following commands in interactive SQL:

```
MODIFY GENUSTYWGT TO TRUNCATED;
COPY TABLE GENUSTYWGT (GENUSNAM=C6, DUM1=D1, NTLINDVAL=C2,
      DUM2=D1, ORDNO=C2, DUM3=D1, RLTYP=C0NL) FROM
      "(device and directory specification in PFNAMEF):GENUSTYWG.IN";
MODIFY GENUSTYWGT TO ISAM ON GENUSNAM, NTLINDVAL, ORDNO;.
```

(Program GENGENUST could also be revised to contain embedded versions of these commands, but it could then be run only by the database administrator.)

4. Program GENIXRTYP ("generate table INDXRTYP")

From the information on actual resource types in INGRES table GENTYP0, Program GENIXRTYP prepares a file with the appropriate information for (the first three columns of) INGRES table INDXRTYP. Thus, like Program GENGENUST, Program GENIXRTYP can be useful when the actual resource types, as specified in table GENTYP0, change. Table IV-5 indicates some features of Program GENIXRTYP.

Table IV-4. SELECTED INFORMATION ABOUT AUXILIARY PROGRAM GENGENUST

Purpose: From the information on actual resource types in INGRES table GENTYP0, prepares a file with the appropriate information for (the first four columns of) INGRES table GENUSTYWGT (useful when actual resource types, as specified in table GENTYP0, change).	
ASCII files read:	None
INGRES tables accessed for retrieval:	PFNAMEF GENUSINFO GENTYP0 LIMVINFO (column dimval)
INGRES tables updated:	GENUSINFO (column nrltyp) INDXBINFO (column ivalrl, for single-resource-category indices)
ASCII files output:	GENUSTYWG.IN--output file containing revised contents for INGRES table GENUSTYWGT Logical unit 6--informative messages
Subroutines:	None

Table IV-5. SELECTED INFORMATION ABOUT AUXILIARY PROGRAM GENIXRTYP

Purpose: From the information on actual resource types in INGRES table GENTYP0, prepares a file with the appropriate information for (the first three columns of) INGRES table INDXRTYP (useful when actual resource types, as specified in table GENTYP0, change).	
ASCII files read:	None
INGRES tables accessed for retrieval:	PFNAMEF INDXBINFO INDXGTYP (view)
INGRES tables updated:	INDXBINFO (column ivalrl, for compound resource-based indices)
ASCII files output:	INDXRTYP.IN--revised contents for INGRES table INDXRTYP GENIXRTYP.WAS--file of informative messages
Subroutines:	None

Program GENIXRTYP operates as follows. From table INDXBINFO, it retrieves the information about each index, in turn. If the index code number is greater than 10, or if the value in column nsteps is 1, no further processing is done for that index. Thus processing is done only for the compound-resource-based indices, which have code numbers 4 or 6. (A review of Chapter II, Section B.2, might be helpful.) For each compound-resource-based index, the program performs an iterative selection from INGRES table INDXGTYP. This table is a view, joining information from INGRES tables INDXGENUS and GENTYP0 (see Reference [5], Chapter 15, for information about views). Table IV-6 shows the columns of table INDXGTYP and their meanings. For details of the algorithm, the reader is referred to the GENIXRTYP computer code. The end result is the union set of actual resource types for that index (as described in Chapter II, Section B.3.b). The union set information is written on the output file INDXRTYP.IN, which resides in the device and directory specified in INGRES table PFNAMEF. Column ivalrl of table INDXBINFO is updated to the number of actual resource types in this union set.

After Program GENIXRTYP has been run, the database administrator should change the contents of INGRES table INDXRTYP. This can be done by executing the following commands in interactive SQL:

```
MODIFY INDXRTYP TO TRUNCATED;
COPY TABLE INDXRTYP (INDXNAM=C6, ORDNO=C3, DUM1=D1,
   RLTYP=C0NL)
   FROM "(device and directory specification in PFNAMEF):INDXRTYP.IN";
MODIFY INDXRTYP TO ISAM ON INDXNAM, ORDNO;.
```

(Program GENIXRTYP could also be revised to contain embedded versions of these commands, but it could then be run only by the database administrator.)

5. Program GENEFFDAT ("generate effectiveness data lines")

Program GENEFFDAT prepares shell files of more disaggregated data lines (with zeros in the space for the data value), for all the effectiveness variables, organized by category. (The reader may wish to review Chapter I, Section B, and Chapter II, Section D, for information on the format and use of these data lines.) When actual resource types change, Program GENEFFDAT should be run to ensure that all of the necessary data lines

Table IV-6. COLUMNS OF INGRES TABLE INDXGTYP AND THEIR MEANINGS

Column Name	Type and Length	Meaning
indxnam	character 6	name of index
istep	integer 1	step number (of resource category within index; as in table INDXGENUS)
genusnam	character 6	code name of resource category for above step
ordno	integer 2	order number of actual weapon type in this resource category
rltyp	character 10	name of actual weapon type in this resource category

are present in the files of more disaggregated data.[9] Table IV-7 indicates some features of Program GENEFFDAT.

Program GENEFFDAT has a subroutine, named GNDTSB (generate data subroutine), which generates (and prints out) all the data lines for one effectiveness variable (the name of which is passed into the subroutine). One call to GNDTSB is made for each effectiveness variable. Section 5.b discusses Subroutine GNDTSB; Section 5.a discusses the output file structure of Program GENEFFDAT and the structure of the different calls to GNDTSB.

Subroutine CHRINI, as described in Chapter III, Section I.1, is also used by Program GENEFFDAT; one call is made to it at the beginning of the program. After calling Subroutine CHRINI, Program GENEFFDAT retrieves the device and directory specification listed in INGRES table PFNAMEF. All output files will be put in this directory.

[9] Throughout this section and Sections 6 and 7, the phrase "data line" will denote a line with a variable name, a sequence of component names (one for each dimension of the variable), with '0.00000' in the place of a data value. The program ensures that the data lines are in the correct format to be accessed by the aggregator program itself (see Table II-23).

Table IV-7. SELECTED INFORMATION ABOUT AUXILIARY PROGRAM GENEFFDAT

Purpose: Prepares shell files of more disaggregated data lines (with zeros in the space for the data value), for all the effectiveness variables, organized by category.		
ASCII files read:	None	
INGRES tables accessed for retrieval:		
	EVARINFO GENTYP0 INDXBINFO INDXDTYP INDXGENUS	INDXRTYP PFNAMEF RVARINFO SPLITINDG SPLITINDX
INGRES tables updated:	EVARINFO (columns filname, ndatlin)	
ASCII files output:	Logical unit 6--output of error messages GC--.DAT--files of data lines GC--.WAS--files of informative messages	
Subroutines:	GNDTSB CHRINI BADSHO	

a. Output File and Category Structure

Recall from Reference [15] and Appendix F of Reference [2] that a category has been assigned to every NAVMOD input. This category is identified by a combination of a category name and category number. The category name and number of each NAVMOD input are stored in INGRES table EVARINFO (see Table II-18).[10] Program GENEFFDAT retrieves the distinct category name and number combinations from table EVARINFO and stores them in a working array. The following procedure is performed for each category that contains effectiveness variables.

First, an output file for the data lines is opened. The file name is based on the category name and number; specifically, the file name is a concatenation of the characters 'GC', the first four letters of the category name, and the character-encoded version of the

[10]The reader should be aware of the special categories GENRAL.4, the ordnance capacity inputs, as listed in Reference [15], and MULTPL.0, the effectiveness variables that fit into two or more of the categories listed in Reference [15]. Files GCGENR04.DAT and GCMULT00.DAT are created to hold data lines for variables in these two categories, respectively.

category number as determined by Subroutine CHRINI. The file extension is '.DAT'. With one exception, to be discussed presently, this output file will contain the data lines for all of the effectiveness variables in this category. A waste file, with the same file name and the extension '.WAS', is created to hold informative messages. The files reside in the device and directory specified in table PFNAMEF.

Then, all of the variables (variable names) in this category are retrieved from table EVARINFO and stored in the working vector VNAMEV. For each variable:

- column filname of table EVARINFO (in the row for this variable) is set to the name of the output file;
- Subroutine GNDTSB is called to generate the data lines; these are written on the output file;
- column ndatlin of table EVARINFO (in the row for this variable) is set to the number of data lines generated, which has been computed in Subroutine GNDTSB, and is passed back from there (column ndatlin is for informative purposes only; it is not used by the aggregator).

Certain exceptions to this structure are discussed in Section 5.b. When all of the variables in the category have been processed, the output and waste files are closed, and the next category is processed.

b. Subroutine GNDTSB

This subroutine performs the actual generation of the data lines for one variable, whose name is passed into the subroutine and given the local name NAME. The subroutine retrieves the appropriate component names from various INGRES tables and writes out all appropriate combinations of these component names in the correct format (as indicated in Table II-23). This program is similar to Subroutine AGGEFF of the aggregator proper but it is simpler--no operations with weights are performed, only component names are retrieved. (The reader might wish to review the discussion of component names in Chapter II, Section B.4.)

(1) Initial Steps

The first step of the subroutine is to retrieve certain information from INGRES table RVARINFO, for the variable in question, as shown in Table IV-8. If the variable code is not equal to 1, indicating that this is not an effectiveness variable (recall Table II-16), the

Table IV-8. INFORMATION RETRIEVED FROM INGRES TABLE RVARINFO BY PROGRAM GENEFFDAT

Column of RVARINFO	Brief Description[a]	Associated FORTRAN Variable
vnamer	Variable name	NAME
vcoder	Variable code number	IVCODE
nodr	Total number of dimensions (NAVMOD plus added)	NODR
nfn	1 if variable is integer; 2 if real	NFN
indx1	Name of first index	INDX(1)
indx2	Name of second index	INDX(2)
indx3	Name of third index	INDX(3)
indx4	Name of fourth index	INDX(4)
indx5	(blank)	INDX(5)
indx6	(blank)	INDX(6)

[a]For fuller descriptions of the columns of table RVARINFO, see Chapter II, Section C.1.b.

subroutine ends. Otherwise, the subroutine initializes to all blanks a large character-length-10 matrix named CMPMTX, which is used to store the component names as they are retrieved from INGRES tables. A few other working variables are also initialized.

If the variable contains an adaptive index, the procedure is somewhat more complicated than if it does not. Section b.2) therefore explains the basic data line generation algorithm, used for variables without an adaptive index, and Section b.3) discusses the modifications that handle the case of an adaptive index. This separation of the two discussions is useful for explaining the algorithm, but does not correspond exactly to the computer code. It is hoped that this discussion, along with examination of the computer code, will lead to sufficient understanding of the subroutine.

(2) Data Line Generation If There Is No Adaptive Index

In this case, the array element CMPMTX(I,J) represents the I-th component name of the J-th index, where J ranges from 1 through NODR, the number of dimensions of the

variable as it is used in the more disaggregated data (see Table IV-8). For a given J, I ranges from 1 through the appropriate number of component names for the J-th dimension. This number is indicated by the working variable LIMU(J). Array CMPMTX is dimensioned as 99 by 6; for J such that NODR < J ≤ 6, LIMU(J) is set to 1 and CMPMTX(1,J) remains blank. Otherwise, the generation algorithm determines the values of LIMU(J) and CMPMTX(I,J) by retrieval from the INGRES database. The specifics depend on the index code and other information in table INDXBINFO. The description below is fairly thorough, but for precise information, the reader is referred to the computer code. The reader might also wish to review the information on taxonomy of indices, in Chapter II, Section B.2.

The following procedure is performed for each value of J between 1 and NODR, inclusive. As shown in Table IV-8, INDX(J) denotes the name of the J-th index. The program retrieves certain information about this index from INGRES table INDXBINFO, as shown in Table IV-9, which is similar to Table III-7. (If at any point in the procedure, the program fails to find requested information in the database, Subroutine GNDTSB calls Subroutine BADSHO, which prints a message and stops the program. This signifies an error in the component name data in the database. The file of informative messages can be used to pinpoint where the program stopped; this might aid in fixing the error.)

If the J-th index is a detail (code 30) index, the program retrieves the component names from table INDXDTYP (for the index currently being considered). If the J-th index is a single-resource-category index (code 20, or code less than 10 with NSTEPS=1), the program retrieves the component names from table GENTYP0, for the (one) resource category associated with the index. This resource category is given by the working variable STPGEN (see Table IV-9). If the index is numerical (code 12, 14, or 16), the component names are character-encoded integers, from 1 through the upper limit MXVAL (see Table IV-9). If the index is compound and resource-based (index code less than 10 with NSTEPS not equal 1), then the component names are retrieved from table INDXRTYP.

In all of these cases, as they are retrieved or generated, the component names are assigned to elements CMPMTX(I,J), for successive I; the names are also counted and the count is assigned to LIMU(J). (In the case of a numerical index, LIMU(J) is set equal to MXVAL.)

Table IV-9. INFORMATION RETRIEVED FROM INGRES TABLE INDXBINFO BY PROGRAM GENEFFDAT

Column of INDXBINFO	Brief Description[a]	Associated FORTRAN Variable[b]
indxnam	Name of Jth index of variable under consideration	INDX(J)
indxcode	Index code number	IXCODV(J)
ivalrl	Number of actual resource types associated with this index	IVALRL
mxvalntl	Upper limit on number of notional resource types or numerical upper limit	MXVAL
nsteps	Number of steps for this index	NSTEPS
genus	Single resource category, if any, associated with this index	STPGEN

[a] For fuller descriptions of the columns of table INDXBINFO, see Chapter II, Section B.3.a.
[b] "J" represents the J-th index or dimension of the variable under consideration.

After the values of CMPMTX(I,J) and LIMU(J) have been determined for all relevant I and J, Subroutine GNDTSB determines the total number of data lines for the variable currently being processed as

$$\text{NLINNU} = \prod_{J=1}^{\text{NODR}} \text{LIMU(J)}.$$

If this number does not exceed certain fixed value (4000 is coded in the current version of the program), the data lines will be written onto the output file established for the variable's category, as explained in Section 5.a. If this limit is exceeded, the subroutine creates a special output file for the data lines associated with the variable. This file resides in the directory specified in INGRES table PFNAMEF. Its extension is 'DAT', and its file name is constructed as a concatenation of the characters:

- 'GO'
- the letters of the name of the variable; and
- right-padding with zeros, as necessary to reach a total of eight characters.

If this special file is created, the entry in column filname of INGRES table EVARINFO, in the row for the variable in question, is updated to the name and extension of this file.

The actual data lines are then generated and written on the output file, whatever file this might be. A nest of six DO loops is used; the j-th loop runs from 1 to LIMU(j), has index IAj, and runs over the component names for the j-th index, i.e., entries CMPMTX(IAj,j). Within the nest, exactly one tuple

(IA1, IA2, IA3, IA4, IA5, IA6)

is specified. The corresponding set of component names

(CMPMTX(IAj,j) | j=1,6)

is written on the output file, along with the variable name and the number '0.00000', in the format specified in Table II-23. Recall that for j > NODR, IAj assumes only the value 1, and the value CMPMTX(1,j) is blank. (Values for j=5 and j=6 are currently always blank and are written in A2 format.)

(3) Modifications for Treatment of Adaptive Indices

The reader might wish to review the basic information on adaptive indices presented in Chapter II, Section E. For each index, adaptive or not, the information shown in Table IV-9 is retrieved from table INDXBINFO. For those J such that the J-th index is not adaptive and is not the base index associated with an adaptive index, the meaning and computation of the working variables LIMU(J) and CMPMTX(I,J) is exactly as described in the preceding paragraphs.

Suppose that the d-th index of the variable under consideration is adaptive (i.e., its index code is 10). Using table SPLITINDG, the subroutine determines the order number, d*, of the base index associated with this adaptive index. The variable names for d and d* in the computer code are KEYER and KEYEE, respectively. The number of steps associated with the index is retrieved from table INDXBINFO and is assigned to the working variable MSTEP. For each value of ISTEP from 1 through MSTEP, the name of the resource category associated with the ISTEP-th step of the adaptive index is retrieved from table SPLITINDX and is assigned to the working variable GSVEC(ISTEP) (this is a character-length-6 vector). If the base index is resource-based (i.e., non-numerical), the

name of the resource category associated with the ISTEP-th step of the base index is retrieved from table INDXGENUS and assigned to the working variable GKVEC(ISTEP).

The nest of six DO loops described above for generation of data lines is itself nested within a loop on step, which goes from 1 through MSTEP. (For variables without an adaptive index, MSTEP is set to 1.) On pass ISTEP of this loop, the working variables LIMU(d), LIMU(d*), CMPMTX(i,d), and CMPMTX(i*,d*) are recomputed (for the given d and d*, for i from 1 through LIMU(d), and for i* from 1 through LIMU(d*)) and data lines are written using the revised LIMU and CMPMTX arrays. This is done as follows.

The component names CMPMTX(i,d) for the ISTEP-th step of the adaptive index are retrieved from table GENTYP0, for resource category GSVEC(ISTEP); the number of such names is determined during the retrieval and is assigned to LIMU(d). If the base index is resource based, a similar procedure is performed for the base index, i.e., the component names CMPMTX(i*,d*) for the ISTEP-th step of the base index are retrieved from table GENTYP0, for resource category GKVEC(ISTEP); the number of such names is determined during the retrieval and is assigned to LIMU(d*). If the base index is numerical, LIMU(d*) is set to 1 and CMPMTX(1,d*) is set to the character-encoded value of ISTEP (i.e., CHRINT(ISTEP)).

The total number of more disaggregated data lines associated with the variable is computed as

$$\text{NLINNU} = \sum_{\text{ISTEP}=1}^{\text{MSTEP}} \prod_{\text{J}=1}^{\text{NODR}} \text{LIMU(J)}$$

with the values of LIMU recomputed as appropriate for step ISTEP. If the variable contains an adaptive index, no special file is opened if NLINNU is larger than the cutoff value; all data lines are written to the output file corresponding to the variable's category.

6. Program GENROLDAT ("generate resource, ordnance, and limit variable data lines")

Program GENROLDAT prepares a shell file of more disaggregated data lines (with zeros in the space for the data value), for the resource, ordnance, limit, and allocation fraction variables. Table IV-10 indicates some features of Program GENROLDAT.

Table IV-10. SELECTED INFORMATION ABOUT AUXILIARY PROGRAM GENROLDAT

Purpose: Prepares a shell file of more disaggregated data lines (with zeros in the space for the data value), for the resource, ordnance, limit, and allocation fraction variables.	
ASCII files read:	None
INGRES tables accessed for retrieval:	PFNAMEF RVARINFO INDXBINFO GENTYP0
INGRES tables updated:	None
ASCII files output:	Logical unit 6--terminal output of error messages GCROL000.DAT--main output file; contains data lines GCROL000.WAS--waste file of informative messages
Subroutines:	CHRINI BADSHO

The program starts by calling Subroutine CHRINI, exactly as Program GENEFFDAT does (see Section 5.a and Chapter III, Section I.1). Program GENROLDAT then retrieves the device and directory name stored in INGRES table PFNAMEF. Then, the program opens an output file, named GCROL000.DAT, to hold the data lines. An output waste file, GCROL000.WAS, is opened to hold informative messages. Both of these files reside in the directory specified in INGRES table PFNAMEF. (The letters ROL are mnemonic for resource, ordnance, and limits.)

After that, the program selects all variables from table RVARINFO with variable code numbers greater than 1, less than 20, and not equal to 7. As Tables II-16 and II-17 indicate, these code numbers correspond to resource, ordnance, limit, and allocation fraction variables used in the more disaggregated data. For each such variable, a data line generation procedure is performed. This procedure is nearly the same as Subroutine GNDTSB of Program GENEFFDAT, as described in Section 5.b, but it is located within the main program GENROLDAT itself, rather than being a separate subroutine. Other major differences from Subroutine GNDTSB are that a small block of code exists to handle code-40 indices, which are used only in Program GENROLDAT (as the second indices of allocation fraction variables). In this case, an upper limit is set as the value in column

mxvalntl of table INDXBINFO, in the row for the index in question. The working variable LIMU(2) is set to this upper limit. The component name CMPMTX(I,2) is set to the character-encoded integer I, for values of I from 1 through the upper limit. Another difference is that there is no code to handle adaptive indices, as these may be used for effectiveness variables only.

If data are not found during execution, Subroutine BADSHO operates in the same manner as it does in Subroutine GENEFFDAT. Otherwise, after all of the relevant variables have been processed, the output files are closed.

When data values are edited into the data lines, the user should remember that the row sums of the allocation fraction variables should equal unity (see Chapter II, Section C.3.c).

7. Program CHGEFFDAT ("change effectiveness data lines")

Program CHGEFFDAT prepares a shell file of data lines for each effectiveness variable on an input list. As discussed in Section D.2.c, this program can be useful when the added indices for one or a few effectiveness variables are changed. Table IV-11 indicates some features of Program CHGEFFDAT.

The program starts by calling Subroutine CHRINI, exactly as Program GENEFFDAT does (see Section 5.a and Chapter III, Section I.1). Program CHGEFFDAT then retrieves the device and directory name stored in INGRES table PFNAMEF.

The program then enters a loop; on each pass of the loop, it reads in a variable name from the file associated with logical unit 3 (which can be assigned to the terminal if interactive input is desired). The program will stop if it encounters the character string 'ZZZZZZ' or an end-of-file on unit 3. For each variable that is input, the following procedure is performed.

The program opens an output file for the data lines associated with the variable. This file's extension is 'DAT', and its file name is constructed as a concatenation of the following characters: GS, the letters of the name of the variable, and right-padding with zeros, as necessary, to reach a total of eight characters. A waste file, with the same file name and the extension '.WAS', is opened to hold informative messages related to this variable. Both of these files reside in the directory specified in INGRES table PFNAMEF.

Table IV-11. SELECTED INFORMATION ABOUT AUXILIARY PROGRAM CHGEFFDAT

Purpose:	Prepares a shell file of more disaggregated data lines (with zeros in the space for the data value), for each effectiveness variable on an input list.
ASCII files read:	Logical unit 3, a list of names of input variables to be processed; can be assigned to terminal
INGRES tables accessed for retrieval:	EVARINFO, GENTYP0, INDXBINFO, INDXDTYP, INDXGENUS, INDXRTYP, PFNAMEF, RVARINFO, SPLITINDG, SPLITINDX
INGRES tables updated:	None
ASCII files output:	Logical unit 4--prompts; should be assigned to terminal GS--.DAT--file of data lines for a variable GS--.WAS--file of informative messages for a variable
Subroutines:	GNDTSB CHRINI BADSHO
Comment:	Depending on the particular variables processed, some of the INGRES tables listed above might not be accessed.

One file of data lines and one waste file are created for each variable on the input list. Subroutine GNDTSB is then called to generate the data lines. This subroutine is similar to Subroutine GNDTSB of Program GENEFFDAT (as described in Section 5.b); the major difference is that no additional file is opened if the number of data lines for the variable is large.

Program CHGEFFDAT generates new shell data lines for the variables entered as input to it. After the program has been run, the user should update the main files that contain the more disaggregated data for these variables. For each variable processed by CHGEFFDAT, the user should locate the file that contains the data for the variable (column filname of table EVARINFO), delete the old data lines for the variable from this data file (use a text editor), introduce the newly generated shell file GS--.DAT into the data file, and edit appropriate new data values into the shell data lines.

8. Program GENREPORT ("genus report")

Program GENREPORT prepares a report of the resource categories and the actual resource types in them. Its operation is simple. First, it retrieves, from table GENUSINFO, information on each resource category. For each resource category, in turn, the actual resource types corresponding to this category are retrieved from INGRES table GENTYP0, and are written out, in a formatted manner, on file GENREPORT.OUT, which resides in the directory from which the program is being run. Table IV-12 indicates some features of Program GENREPORT.

Table A-8 of Appendix A is similar to the output of Program GENREPORT, for the actual resource types currently in the database.

9. Program DIMREPORT ("dimension report")

Program DIMREPORT prepares a report showing definitions of NAVMOD input variables (as used in NAVMOD) and definitions of their NAVMOD and added indices. The added indices are the ones currently specified in INGRES table RVARINFO. The user can change the added indices for effectiveness variables; Program DIMREPORT can be useful in indicating the current status of these. Table IV-13 indicates some features of Program DIMREPORT.

Some of the source code of Program DIMREPORT is located on included files. (The purpose of these included files is to have a list of NAVMOD inputs available to the program.) The information file BLINKP.DAT read by the aggregator is one of these files; the other included files of DIMREPORT are also used as included files in the aggregator. If some change in the aggregator necessitates the running of Program CHGINP3, then Program DIMREPORT should be recompiled, as some of the output files of CHGINP3 will be included files in DIMREPORT.

Program DIMREPORT operates on each NAVMOD input, in turn, as follows. First, the information on that variable is retrieved from INGRES table RVARINFO. Then, the definition is retrieved from INGRES table VARDEFS, and is printed on the output file (which is located in the directory from which the program is being run). Table VARDEFS is used by this auxiliary program but is not used by the aggregator itself. It is used by some of the utilities in the interaction selector preprocessor (to display a variable definition on the screen). The texts of the definitions are the same as those in Appendix F of

Table IV-12. SELECTED INFORMATION ABOUT AUXILIARY PROGRAM GENREPORT

Purpose: Prepares a report of the resource categories and the actual resource types in them.	
ASCII files read:	None
INGRES tables accessed for retrieval:	GENUSINFO GENTYP0
INGRES tables updated:	None
ASCII files output:	GENREPORT.OUT--file containing the report
Subroutines:	None

Table IV-13. SELECTED INFORMATION ABOUT AUXILIARY PROGRAM DIMREPORT

Purpose: Prepares a report showing definitions of NAVMOD input variables (as used in NAVMOD), and definitions of their NAVMOD and added indices.	
ASCII files read:	None
INGRES tables accessed for retrieval:	RVARINFO INDXMSG VARDEFS
INGRES tables updated:	None
ASCII files output:	DIMREPORT.OUT, a large file which contains the output report
Subroutines:	None (does have block data routine)
Included Files:	BLKINP.DAT CMISUM.PAR IVARB.CMN

Reference [2]. The names of the different indices for the variables have been retrieved from table RVARINFO. For each index, in turn, the program retrieves its definition from table INDXMSG and prints the index name and definition on the output file. Notional indices and added indices (if any) are listed separately.

D. IMPLEMENTING SELECTED TYPES OF PROGRAM AND DATABASE CHANGES

As mentioned in the introduction to Chapter IV, this section addresses some of the issues that arise in changing the aggregator to be consistent with some of the more common NAVMOD or database changes that might be desired. Depending on the particular change, modifications to any or all of the aggregator computer source code, the database NAVPRE, and the input data files might be necessary.

The reader is assumed to be familiar with the database structure and the aggregator computer program, as explained in Chapters II and III. Some examination of the aggregator computer code might help in implementing certain of the changes. Certain of the changes discussed here can be implemented in an essentially automated manner, by running one of the auxiliary computer programs. Other changes require manual update (via QBF or SQL) of the contents in the NAVPRE database or require that data files be edited with a text editor. Thus different types of changes require different levels of familiarity with NAVMOD, the aggregator program, and the database.

Section D.1 discusses aggregator changes that become necessary because of changes in the NAVMOD inputs. A number of different types of changes in the NAVMOD inputs are considered. Section D.2 discusses some additional types of aggregator changes that might be desired. These comprise:

- changing the actual types of resources within a category;
- changing the combat-related names on a "detail" index;
- changing the added indices of effectiveness variables; and
- inventing new indices, to be used for added indices.

1. Changes to NAVMOD Inputs

a. General Remarks

This section examines how to change the aggregator if changes are made to the file DCOMMN.DAT, which contains information about the NAVMOD input variables and is

discussed in Appendix C of Reference [2]. (A review of that appendix might be helpful.) The purpose of the aggregator is to produce a set of values for the NAVMOD input variables; the computations of the NAVMOD model are not directly relevant to the aggregator. Thus, the only changes to NAVMOD examined here are changes to the NAVMOD inputs. Recall that changes to the NAVMOD inputs that involve addition or redimensioning of a NAVMOD input require that the file DCOMMN.DAT be edited in an appropriate manner and an auxiliary program be run, as discussed in Appendix C of Reference [2]. This is necessary for NAVMOD's input routine to work correctly with the new input structure. In sum, most major changes to the NAVMOD inputs are reflected by changes to DCOMMN.DAT. We thus restrict the discussion to changes to DCOMMN.DAT that affect the NAVMOD inputs.[11,12]

All of the changes described in this section involve, among other things, changes to the DCOMMN.DAT file. If any change--including any of the changes described in this section--is made to DCOMMN.DAT, then the following procedures must be performed (additional procedures that depend on the particular change might also have to be performed):

- The auxiliary program CHGINP3 should be run. (The output files of this program are included files or input files for NAVMOD and/or the aggregator.)
- Recompile and relink NAVMOD, as indicated in Appendix C of Reference [2].
- Recompile and relink the aggregator preprocessor computer program source code (rerunning through the Embedded SQL FORTRAN precompiler is not necessary).
- Program CHGINP3 generates a new version of the information file BLKINP.DAT. This new version should be used as the input file for the revised NAVMOD and aggregator programs (see Chapter I, Section C.4.a).

During the recompilation processes, the user should make sure that the appropriate files generated by Program CHGINP3--the included files for NAVMOD and the aggregator--reside in the same directories as the main source code files for these programs (copy the files as necessary). Reference [2] (Chapter III, Section D) discusses the included files for

---

11 It is conceivable that a NAVMOD input is not redimensioned (i.e., DCOMMN.DAT does not change) but that the input's use in NAVMOD is changed so much that some special restriction of the sort described in Section B, above, becomes applicable to it. Technically, this would be a "change in a NAVMOD input that would affect the aggregator," but we will not deal with such second-order considerations in this section.

12 Some changes to the dimensioning of NAVMOD working variables stored in COMMON blocks also necessitate changes to DCOMMN.DAT, but this doesn't affect the aggregator. This section will deal with changes to NAVMOD input variables only.

NAVMOD, and Table A-5 of Appendix A lists the included files for the aggregator program.

The changes to the aggregator computer program necessitated by changes to DCOMMN.DAT consist only of the revised included files and the revised BLKINP.DAT file. (These files relate to the storage and output of the aggregated values.) The remainder of the necessary changes to the aggregator consist of changes to the contents of the database NAVPRE.

As indicated in Appendix C of Reference [2], merely changing the values of the user-changeable symbolic constants in NAVMOD is a simpler procedure than adding, deleting, redimensioning, or changing the COMMON locations of NAVMOD inputs. Therefore, these two procedures are discussed in separate subsections.

b. Resetting the Values of NAVMOD User-Changeable Symbolic Constants

(1) Introduction

Recall that the values of certain of the symbolic constants in NAVMOD can be changed by the user (the values of the other symbolic constants are fixed or are determined by formula). The procedure for adjusting the aggregator when the values of (any of) the user-changeable symbolic constants are changed is as follows:

1) Edit the file DCOMMN.DAT in accordance with the procedure described in Section B of Appendix C of Reference [2]. Be sure to follow the caveats listed in that section.
2) Run Program CHGINP3 and recompile NAVMOD and the aggregator, as discussed in Section 1.a. This step can be performed after or concurrently with the other steps below.
3) Run auxiliary program GENMXVAL, which will update the database NAVPRE to reflect the new symbolic constant values. (The person running the program must have update permission on certain tables in the database; see Table IV-3.)
4) Run auxiliary program GENGENUST. This is necessary because the entries in column ntlindval of table GENUSTYWGT should assume values up to the value of some symbolic constant, which might have been changed.
5) The database administrator should update the contents of table GENUSTYWGT as discussed in Section C.3.
6) Depending on the specific changes to the symbolic constants, the files of more disaggregated data may need to be changed. Programs GENROLDAT, GENEFFDAT, and/or CHGEFFDAT might be helpful in generating any additional data lines needed.

To discuss the necessary changes to the files of more disaggregated data, let us first examine the user-changeable symbolic constants more closely. Examination of Tables III-3, III-4, and Figure C-1 of Reference [2], and comparison of those tables and figures with Table II-13 of the current paper reveals that the user-changeable symbolic constants fall into the following groups:

- MXABSM, MXKBD, MXKBES, MXKRB, and MXPPSM are single-resource category (code 2) indices
- MXKRSS is a special case; it is a compound resource-based (code 6) index, but it is related to the single resource category index MXKRSN; a change in value for MXKRSS yields a change in value for MXKRSN (see the discussion in the description of Program GENMXVAL, in Section C.2)
- MXLOC is a numerical (code 12) index that is used as a dimension for many NAVMOD inputs
- MXMPSG, MXNMP, MXSAG, and MXTC are also code 12 indices, but are used as dimensions for relatively few NAVMOD inputs
- MXTABA, MXTABD, MXTABH, MXTABP, MXTABS, and MXTABV are code 12 indices, are used as dimensions for relatively few NAVMOD inputs, but have special treatments (as described in Section B).

As indicated in Reference [2], Chapter III, Sections B and C.4, each of these symbolic constants has an associated limit variable. These limit variables are used in the aggregator, appearing in table LIMVINFO and in the file of more disaggregated data for resource, ordnance, limit, and allocation fraction variables. Changes to the symbolic constant values are often made because it is desired to set the associated limit variable to a value larger than the original value of the symbolic constant. This implies that the aggregator program will search for additional lines of more disaggregated data. These lines must thus be added to the appropriate data files. No fully automated method has (currently) been developed to do this. The remainder of this subsection discusses the types of data changes that will need to be made, and suggests ways of dealing with them.

In the remainder of this section, the term "data changes" will refer to changes to the files of more disaggregated data, not changes to the contents of the database NAVPRE, which are assumed to have been made as indicated above. Even if a symbolic constant value changes, data changes are unnecessary unless the value of the corresponding limit variable also changes, so assume this latter case holds. The new value should be edited into the data line for the limit variable. (This data line appears in the file of more disaggregated data for resource, ordnance, limit, and allocation fraction variables.) The following subsections indicate the other changes that would be necessary.

(2) Changes to Symbolic Constants That Correspond to Resource-based Indices

In addition to an associated limit variable, each symbolic constant in this group has an associated resource category and allocation fraction variable. Table IV-14 shows the specific correspondence. The discussion here is in terms of one such symbolic constant. Changes to different such symbolic constants operate independently of one another. No changes to the effectiveness variables need be made.

For indices in this group, a change in the limit variable means a change in the number of notional resource types to be considered. The values of the corresponding allocation fraction variable should be changed to reflect appropriate assignments of actual resource types to the notional resource types. (The reader might wish to review the discussion of allocation fraction variables in Chapter II, Section C.3.c.) The specific types of changes necessary depend on the type of change in the limit variable, as follows.

If (the symbolic constant value and/or) the limit variable value has decreased, no new data lines for the allocation fraction variable are needed, but the values on the existing data lines should be changed to ensure that no actual resources will be assigned to notional resource types that are no longer being considered.

If the limit variable value has increased beyond the old symbolic constant value, new data lines for the allocation fraction variable will be necessary. In these data lines, the second component would be a character-encoded integer; the integer values would range from 1 plus the old symbolic constant value up to the new limit variable value.

If the symbolic constant value has increased, and the limit variable has increased, but not beyond the old symbolic constant value, then Program GENROLDAT would have generated data lines for the allocation fraction variable that could accommodate values for the additional notional resource types to be considered now. These data lines would not have been used previously; now, the appropriate data values should be edited into them.

In all of these situations, making the appropriate data changes with a text editor would probably not be too time consuming.

Some remarks about the symbolic constant MXKRSS are in order. The value of MXKRSS is an upper limit on the number of types of Red surface ships in regions. These types encompass an input number of types of non-resupply ships (this number is given by

Table IV-14. USER-CHANGEABLE NAVMOD SYMBOLIC CONSTANTS THAT CORRESPOND TO RESOURCE-BASED INDICES, WITH ASSOCIATED INFORMATION

Symbolic Constant Name	Associated Limit Variable	Associated Resource Category Name	Associated Allocation Fraction Variable	Description of Resource Category
MXABSM	NABSAM	RMABD	FABRSM	Red airbase defense SAMs
MXKBD	NKBDPL	BLFTR	FPLBLB	Blue land-based fighter aircraft
MXKBES	NKBES	BFESC	FBESS	Blue escort ships
MXKRB	NKRB	RLBMR	FATABT	Red bombers
MXKRSS[a]	NKRSN	RFNRP	FRSFNR	Red non-resupply surface ships in regions
MXPPSM	NPPSAM	RMPPD	FPPRSM	Red power-projection-defense SAMs

[a]See the discussion at the end of this subsection.

the value of the limit variable NKRSN) plus one type of resupply ship. NAVMOD can only model one type of Red resupply ship (in regions; NAVMOD can also model Red barrier tenders). Thus, a change in the value of MXKRSS should be interpreted as a desire to change the number of types of non-resupply ships to be considered. The values of the limit variable NKRSN and allocation fraction variable FRSFNR should then be changed appropriately (see Chapter II, Section C.3.b). Also, in DCOMMN.DAT, MXKRSS should be given a value of at least 2, to encompass at least one type of non-resupply ship.

(3) User-Changeable Symbolic Constants That Correspond to Numerical Indices

A change in this kind of symbolic constant can affect the data lines for the variables that use this symbolic constant as an index. The component names associated with a code-12 numerical index are character-encoded integers, ranging from one through the value of the limit variable associated with the index. The aggregator will look for data lines with these component names.

Since the index MXLOC is used in so many variables, we suggest a somewhat different treatment for changes to it. Let us first consider the other user-changeable symbolic constants that correspond to numerical indices. As Table IV-15 shows, these indices are used in very few variables.

Table IV-15. USER-CHANGEABLE NAVMOD SYMBOLIC CONSTANTS THAT CORRESPOND TO NUMERICAL INDICES[a]

Symbolic Constant Name	Definition	NAVMOD Inputs Using This Symbolic Constant
MXMPSG	Maximum number of movement periods for any SAG	LSAGMP(MXSAG, MXMPSG) LGSGMP(MXSAG, MXMPSG)
MXNMP	Maximum number of movement periods for task force	LGTHMP(MXNMP) LTFMP(MXNMP)
MXSAG	Maximum number of SAGs	MIMPSG(MXSAG)[b] LSAGMP(MXSAG, MXMPSG) SGSRS(MXKRSS, MXSAG) LGSGMP(MXSAG, MXMPSG)
MXTABA	For discussion of MXTAB symbolic constants, see Section B.2.g.	TAB10T(MXTABA, MZKRB1)
MXTABD		PDINV(MXTABD)
MXTABH		TAB13T(MXTABH, MZKRB1)
MXTABP		PPCVFX(MXTABP)
MXTABS		TAB12(MXTABS)
MXTABV		VTTAB(MXTABV)
MXTC	Maximum number of types of SLOC-delivered cargo.	SLCCS(MZKBS, MXTC)[c]

[a]MXLOC is not included.
[b]The aggregator has a special treatment for this limit variable, as discussed in Chapter II, Section C.3.b.
[c]See Section B.3.c.3).

If the desired new value of the associated limit variable does not exceed the old value of the symbolic constant, Program GENEFFDAT should have generated data lines with the appropriate component names when it was originally run. Additional data values can be edited in as appropriate.

If the desired new value of the associated limit variable exceeds the old value of the symbolic constant, Program CHGEFFDAT can be run on the variables that use the symbolic constant as an index. The file produced by CHGEFFDAT will have the

necessary additional shell data lines. Data values can be edited into these lines, and the lines can then be transferred into the main data file for the variable.

This procedure is also relevant if a change to the symbolic constant MXMPSG is desired, even though the aggregator has a special treatment for the associated limit variable, as discussed in Chapter II, Section C.3.b. (Section B.2.g, above, contains additional information on the treatment of the 'MXTAB_' indices.)

Let us now consider changes to the symbolic constant MXLOC, which indicates region. MXLOC is used as an index in many variables, including resource and ordnance stock variables. Increasing the value of the associated limit variable NLOC can necessitate a considerable amount of new data: all of the data for the additional regions to be simulated. (If, for some reason, MXLOC is changed but NLOC stays the same or is decreased, changes to the data are not necessary, except for checking that the relevant relations in Section B still hold.)

If the desired new value of NLOC is less than or equal to the old value of the symbolic constant MXLOC, Programs GENEFFDAT and GENROLDAT should have generated data lines with the appropriate component names when they were originally run. Additional data values, for the higher-numbered regions, can be edited in as necessary. If the desired new value of NLOC is greater than the old value of the symbolic constant MXLOC, a considerable number of additional data lines will be necessary. Probably the best action to take is to run Programs GENEFFDAT and GENROLDAT again. The data values edited into the old data files would need to be transferred to the new data files. Automated procedures to do this have not currently been developed, but probably could be. The data values for the newly introduced higher numbered regions could then be edited into the new data files by hand.

c. Other Types of Changes to DCOMMN.DAT

As indicated in Appendix C of Reference [2], the use of the DCOMMN.DAT file and the auxiliary CHGINP program allows the NAVMOD user to change the structure of the NAVMOD inputs in certain ways and then to adjust the NAVMOD input routine, in an automated fashion, to be compatible with the new input structure. The types of changes allowed include adding, deleting, or redimensioning NAVMOD inputs, or changing the COMMON blocks in which they appear. If these types of changes to the NAVMOD inputs are made it is also possible, under some circumstances, to adjust the aggregator to function with the new set of inputs. This section discusses these adjustments.

Some of the procedures explained here involve changing the contents of the database; this can be done with QBF (Reference [12]), or SQL (Reference [6]). And as explained in the introduction to Section D.1, any change to the DCOMMN.DAT file means that Program CHGINP3 should be run and that the aggregator program and NAVMOD should then be recompiled. It will be assumed throughout this section that the desired changes to the DCOMMN.DAT file have been made. (Of course, the new input variable structure would reflect changes of the NAVMOD model itself. These changes are assumed to have been implemented in the NAVMOD source code. Also, in changing the structure of the NAVMOD inputs, the caveats listed on pages C-7 and C-8 of Reference [2] should be observed.)

The material of this section applies to changes in effectiveness variables only. If NAVMOD's resource, ordnance and/or limit variables were to change, it probably would be possible to adjust the aggregator accordingly. However, such adjustments might entail extensive changes to the database NAVPRE and the aggregator source code.

The descriptions in Subsections c.1, c.2, and c.3 are in terms of changes to a single variable, but changes to more than one variable can be made at a given time--as far as the aggregator is concerned, changes to different variables operate independently of one another.

(1) Deleting An Input Variable

This change is straightforward: simply remove from tables RVARINFO and EVARINFO the rows for the deleted variable (there should be one such row in each table). If Option 3 or Option 13 is being used (on the input options guide file), the user should check that the newly deleted variable does not appear in the list of variables to be processed (although that condition would not cause an error, merely an informative message).

(2) Redimensioning An Input Variable

Redimensioning here refers to changing the number of dimensions, or the ordering of the dimension indices, of the variable as it is used in NAVMOD. Changes to the added (i.e., non-NAVMOD) indices are discussed in Section 2.c. Merely changing the values of the symbolic constants that indicate dimension limits has been covered in Section 1.

The basic procedure for changing the aggregator when a NAVMOD effectiveness input variable is redimensioned is as follows:

- Change the row for the variable in INGRES table RVARINFO to show the new number of dimensions (column nodn) and sequence of index names (columns indx1 through indx4). (These should be consistent with the change to the entry in the DCOMMN.DAT file for the variable; more on this follows.)
- If the added (non-NAVMOD) indices and/or the total number of dimensions (column nodr) have changed, the revised row of table RVARINO should reflect this also. (Table II-15 shows the meanings of the columns of table RVARINFO.)
- Run the auxiliary program CHFGEFFDAT for the variable. This will yield shell data lines with the proper sequences of component names. (Then edit data values into these new data lines, and replace the old lines in the data file for the variable with the new lines, as indicated in Section C.7.)

In changing the entry of RVARINFO, several caveats should be observed, as follows.

Remember that the variable as used in NAVMOD can have no more than three dimensions and that the total number of dimensions--NAVMOD plus added--cannot exceed four. Observe the caveats on added indices, as stated in Section 2.c, which follows.

When changing the line for the variable in the file DCOMMN.DAT, use existing indices as dimension names as much as possible. That is, if an aggregator index currently exists that is consistent with the use of a new dimension for a variable, use that index. If new indices (symbolic constants, in DCOMMN.DAT) have to be invented, the new indices must be registered in the database NAVPRE in an appropriate manner. Subsection c.5 indicates some issues involved in doing this. For reference, Table II-13 shows the indices currently in the database, with their meanings.

As discussed in Chapter II, Section B.2, most dimension sizes are represented in DCOMMN.DAT by a symbolic constant, but a few are represented by numerical values. For these latter cases, corresponding index names have been used, as shown in Table II-7. Be sure that a numerical value does not appear in the "index" columns of table RVARINFO; use the corresponding index name instead.

If a NAVMOD input has been redimensioned, and one of the changed dimensions has a size represented in DCOMMN.DAT by a numerical value, and if an index corresponding to this numerical value has not been set up in the aggregator, then invent such an index, following the procedure in Subsection c.5, and use this new index in the row in table RVARINFO for the variable.

(3) Adding an Input Variable

If a new effectiveness input variable is added to NAVMOD, the basic procedure for changing the aggregator is similar to the one described in the previous subsection. The following steps should be performed.

- From the line for this variable in the DCOMMN.DAT file, construct the appropriate corresponding sequence of index names for the NAVMOD dimensions (observe the caveats in Subsection c.2).
- Decide on added indices for the variable, as appropriate. Recall that numerical indices should not be used as added indices.
- Add a row to INGRES table RVARINFO for the variable, with the appropriate indexing information. The reader might wish to review Table II-15, which defines the columns of table RVARINFO.
- Add a row to INGRES table EVARINFO for the variable. The reader might wish to review Table II-18, which defines the columns of table EVARINFO. This row should contain the variable name (in column vname), and the name of a file to hold the more disaggregated data (in column filname). The rest of the information in EVARINFO is not necessary to run the aggregator but can be filled in if desired. To rerun Program GENEFFDAT in the future, a category name and number should be specified. The new input most likely will fit into one of the existing categories, as shown in Reference [15].
- Run the auxiliary program CHGEFFDAT for the variable. This will yield shell data lines with the proper sequences of component names. Then edit data values into these data lines, and insert the edited lines into the data file specified in table EVARINFO, for this variable. All of the caveats in Subsection c.2 are applicable here also.

(4) Changing the NAVMOD COMMON Blocks

Among other things, the DCOMMN.DAT file specifies for each NAVMOD input the label of the COMMON block in which values for that input will be stored, when NAVMOD is run. (All NAVMOD inputs must reside in labeled COMMON.) DCOMMN.DAT can be edited to change the COMMON block names, to add new COMMON blocks, and to change the specification of which variables are stored in which COMMON blocks.

This does not have much influence on the aggregator. If Program CHGINP3 is run and the aggregator and NAVMOD are recompiled with the appropriate new included files generated by Program CHGINP3, no problems should occur. To preserve compatibility with the interaction selector preprocessor, the user should update INGRES table

EVARINFO to reflect the new COMMON block assignments for the variables (column cblock; see Table II-18).

(5) New Symbolic Constants in DCOMMN.DAT

As mentioned in Subsection c.2, if a new input variable is added to NAVMOD, it sometimes might be appropriate for one or more dimensions of that variable to be referred to in DCOMMN.DAT by new symbolic constants. Each symbolic constant that is specified in the DCOMMN.DAT file in the dimension declaration of a NAVMOD input variable must be registered as an index in the aggregator; i.e., the database NAVPRE must be updated to reflect this additional index.[13]

It is difficult to give a general procedure for dealing with all types of new indices. In general, the information to be entered into NAVPRE varies quite a bit with the characteristics of the index. Also, a new index sometimes goes along with a new resource category and/or limit variable, and this necessitates considerable update to the database. In a sense, all of the material in Chapters II and III is relevant to the construction of new indices. Further, the fact that a new index is required often means that fairly extensive changes to NAVMOD have occurred.

Subsection c.2 mentioned one specific case where a new index would be necessary: the case in which a NAVMOD input has been redimensioned, and one of the changed dimensions has a size represented in the DCOMMN.DAT file by a numerical value, and an index corresponding to this numerical value has not previously been set up in the aggregator (Table II-7 shows the indices of this type that have been set up so far). In this case, a new index can be registered in the database by the following procedure.

- Invent a name for the new index.
- Add a row to table INDXBINFO with the following information: the entry in column indxnam should equal the new index name, the entry in column indxcode should equal 14, the entries in columns mxvalntl, ivalntl, and nsteps should all equal the numerical value specified in DCOMMN.DAT, and the other columns of table INDXBINFO are irrelevant.
- Update table RVARINFO for each variable that uses the new index, as explained in Subsection c.2. (If the *i*-th dimension of the variable is specified in the DCOMMN.DAT file by the numerical value, then use the new index name in column indx*i* of table RVARINFO, in the row for that variable.)

---

[13]A number of the symbolic constants that appear in the DCOMMN.DAT file are not used in the dimension declarations of NAVMOD inputs, but only those that are so used are relevant to the aggregator.

In general, each index should have exactly one row in table INDXBINFO devoted to it. That is, if a new index is invented, a row should be added to table INDXBINFO. The information in the row should be consistent with Table II-6 and Table II-8. If the index does not involve a new resource category and/or limit variable, then the index is most likely either numerical (code 14), or compound resource-based (code 4 or code 6), reflecting a new combination of existing resource categories. A numerical index can be dealt with as described in the preceding paragraph. If the index is compound resource-based, then: 1) the entry in column nsteps of table INDXBINFO should be set to the number of resource categories in the combination; 2) the individual resource categories should be entered into table INDXGENUS; 3) the auxiliary program GENMXVAL should be run; and 4) the auxiliary program GENIXRTYP should be run and table INDXRTYP should be changed, as described in Section C.4.

2. Aggregator Changes Not Directly Related to NAVMOD Changes

Even if the NAVMOD inputs do not change, the aggregator has several other features under the control of the user. These features are implemented via changes to the database NAVPRE and the files of more disaggregated data. These features all change the component name sequences of certain more disaggregated data elements. This can lead to greater realism in the more disaggregated data.

Four specific types of changes are described here:

- changing the actual types of resources within a category;
- changing the combat-related names on a "detail" index;
- changing the added indices of effectiveness variables; and
- inventing new indices, to be used for added indices.

These are discussed in Sections a through d, which follow. More or less automated procedures, involving the auxiliary programs, have been developed to address the first three topics.

a. Changing the Actual Types of Resources Within a Category

The user can specify the actual types of resources associated with each resource category. The primary INGRES table that shows these resources is table GENTYP0. To register changes in the actual resource types, table GENTYP0 should be edited (with QBF or SQL). However, certain other INGRES tables and more disaggregated data files will

also need to be changed. The relevant INGRES tables can be changed fairly easily, by means of the auxiliary programs. Effecting the appropriate changes to the files of more disaggregated data can be more problematic. For this reason, it is probably best to decide on the actual resource types before data values have been edited into the more disaggregated data lines. It does little harm to list extra actual resource types that one might not want to consider later: by choice of data for the more disaggregated resource variables, it is possible to exclude a listed resource type from the aggregation.

To update the INGRES tables, the following procedures should be performed.

- Edit the appropriate new actual resource types into table GENTYP0, using QBF or SQL. Note that an order number (1,2,3,...) must be assigned to each resource type; see Table II-3. Tables II-1 and A-8 (in Appendix A) might also be helpful.

- Run the auxiliary program GENGENUST. Replace the contents of INGRES table GENUSTYWGT with the output file of GENGENUST and remodify GENUSTYWGT to an ISAM storage structure, as described in Section C.3.

- Run the auxiliary program GENIXRTYP. Replace the contents of INGRES table INDXRTYP with the output file of GENIXRTYP and remodify INDXRTYP to an ISAM storage structure, as described in Section C.4.

The required changes to the more disaggregated data arise because different actual resource types yield different sequences of component names for certain variables. New data values must be determined and the new data lines must be integrated into the existing data files. Automated procedures have not been developed to implement these changes, and a fair amount of editing (with a text editor) might be necessary (some suggestions will be given in the following paragraphs). (As an aside, note that a data file should have at most one line with a given variable name and sequence of component names, otherwise, INGRES errors can occur during program execution.)

To describe the nature of the changes to the more disaggregated data that arise from a change in actual resource types, let us focus on changes to the actual resource types in one given resource category; changes to different resource categories operate independently of one another. The main question is if a certain resource category has had its actual resource types changed, which variables have their more disaggregated data lines (component names) affected by this change? We consider changes to the resource, ordnance, and allocation fraction data, and changes to the effectiveness data separately.

(NOTE: When an instruction is given below to run program GENROLDAT, GENEFFDAT, or CHGEFFDAT, it is assumed that the INGRES database NAVPRE has already been updated as described previously in this section.)

Among the resource, ordnance, and allocation fraction variables, the resource variable for the stock associated with the resource category under consideration will certainly be affected by a change in actual resource types, and a reserve resource stock or an allocation fraction variable might also be affected. Table III-5 (table RESGENUS) shows the specific variable names. The other variables in the file of more disaggregated data will not be affected. Program GENROLDAT can be rerun (be careful to save the old version of the data file). The shell data lines for the affected variables can be extracted from the new output file, data values can be edited into them, and the data lines in the old file can be replaced by the newly edited data lines. Or, one need only edit a data value into those shell data lines with sequences of component names that do not appear in the old file (such lines are for the new actual resource types): these lines can then be inserted into the old file. (When determining new values for an allocation fraction variable, the constraints described in Chapter II, Section C.3.c, should be observed.)

An effectiveness variable will be affected by a change in actual resource types if the variable has as one or more of its indices (notional or added):

- a single-resource category index that corresponds to the resource category under consideration (this resource category would be shown in column genus of table INDXBINFO); or
- a compound resource-based index, one of whose component resource categories is the one under consideration (this resource category would be shown in column stepgenus of table INDXGENUS); or
- an adaptive index, where the resource category for one of the steps for this index is the resource category under consideration (this resource category would be shown in column genusstep of table SPLITINDX).

To determine the affected effectiveness variables, the user should first (by searching the tables listed above) determine the indices that fall into the above categories and should then search table RVARINFO to find the effectiveness variables that have such an index.

If relatively few effectiveness variables are affected, Program CHGEFFDAT can be run to generate new shell data lines for these variables; then, new data values can be edited in and the changed data lines can be integrated into the appropriate existing data files. Actually, only the newly generated data lines with new sequences of component names (corresponding to the new actual resource types) need be edited.

If many effectiveness variables are affected, it might be fruitful to simply rerun the entire GENEFFDAT program, and then develop an auxiliary program that will flag the data lines with new sequences of component names (so that the user can enter in a data value) and will simply copy the old data values from data lines whose component name sequences have not changed. Such an auxiliary program could probably be developed fairly easily.

b. Changing the Combat-Related Names in a Detail Index

The combat-related names encompassed by a detail (code 30) index are under the control of the user. If changes in these names are desired, then certain additional changes to the database (table INDXDTYP) and to the more disaggregated data should be performed to maintain data consistency. This section describes these changes, which are relatively straightforward.

The reader might wish to review Chapter II, Section B.2, and Chapter III, Section B, which discuss the motivation behind detail indices and the aggregator's treatment of them.

The following steps should be performed. The following discussion is in terms of a single detail index; changes to the component names of different detail indices operate independently of one another.

- Using QBF or SQL, change table INDXDTYP to reflect the appropriate new combat-related names, order numbers, and relative weights. These should go in columns rltyp, ordno, and rawwgt, respectively, of table INDXDTYP. (The name of the detail index under consideration should appear in column indxnam, in each relevant row.) Table III-2 might be helpful in indicating the format. Note that an order number (1,2,3,...) must be assigned to each combat-related name. The relative weights need not sum to 1--Subroutine STINDD of the aggregator performs the necessary normalization.
- Determine (via a search of table RVARINFO) which variables use the changed index (as an added index). Run Program CHGEFFDAT for these variables. Edit in the appropriate new data values in the shell data lines output by CHGEFFDAT and replace the corresponding old data lines in the files of more disaggregated data, as described in Section C.7.

This procedure can also be used to put additional detail indices into the database. Table INDXDTYP should be edited as described above, with the name of the new detail index in column indxnam. A row should be added to table INDXBINFO, with the name of the index in column indxnam (of table INDXBINFO) and 30 in column indxcode; column ivalrl will be filled in by Subroutine STINDD of the aggregator and the rest of the columns of table INDXBINFO are irrelevant. Table RVARINFO should be edited so that the

desired variables have the new index specified as an added index. Then Program CHGEFFDAT can generate the appropriate new shell data lines.

c. Changing the Added Indices of Effectiveness Variables

As explained in Chapters I and II, the user has considerable freedom to specify the added (i.e., non-NAVMOD) indices of the effectiveness variables. (The reader might wish to review Chapter I, Section B.1.b, for a discussion of the functions added indices can serve.) The database has been set up with a representative selection of added indices. However, if the user wants to specify a new sequence of added indices for an effectiveness variable, the database and more disaggregated data should be changed accordingly. This can be done by the procedure described in the following paragraphs.

Assume that all of the indices in the desired new sequence are indices currently set up in the database (see Table II-13). Section d discusses the establishment of new indices. Recall that for each resource category, the corresponding single-resource-category index is currently registered in the database (with index code 2 or 20), available to be used as an added index, if desired.

The description below is for one effectiveness variable; changes to different effectiveness variables operate independently of one another. Given all the above conditions, the procedure for changing the added index sequence is as follows.

- Using QBF or SQL, change the row of table RVARINFO for the variable under consideration. In particular:

  --change the entry in column nodr to the revised total number of dimensions, if this has changed;

  --keep the first nodn indices as they are (here, nodn refers to the entry in column nodn of the row, the number of dimensions of the variable as it is used in NAVMOD);

  --change the entries in columns INDXi, for nodn $< i \leq$ nodr to the desired new added index names.

- Run program CHGEFFDAT for that variable to generate appropriate new shell data lines. Edit appropriate new data values into these lines and replace the old data lines in the appropriate file of more disaggregated data, as described in Section C.7.

The following caveats should be observed:

--The resource categories should not be changed.

--The notional indices should not be changed (unless NAVMOD variables themselves change, as mentioned in Section 1.c);

--The added indices for the resource and ordnance stock variables should not be changed;

--A numerical index (one with index code 12, 14, or 16) should not be used as an added index.

d. Inventing New Added Indices

In the course of specifying added indices for the effectiveness variables, it might become desirable to invent new indices and to specify them as added indices for certain effectiveness variables.[14]

Each new index must be registered in the database. At a minimum, this implies constructing a new row of table INDXBINFO for the index. In this row, the entry in column indxnam should be set to the name of the new index. The entry in column indxcode should be set to 10 if the index is an adaptive index, 30 if it is a detail index, 4 if it is a compound resource-based index without any associated limit variable, and 6 if it is a compound resource-based index with one or more associated limit variables. (The reader might wish to review Tables II-6 and II-8.) If the index is compound resource-based, the value in column nsteps of the row should be set to the number of resource categories that the index encompasses. The other columns of table INDXBINFO are either irrelevant (for these types of indices) or will be filled in by the aggregator or by certain auxiliary programs.

In addition to table INDXBINFO, certain other INGRES tables need to be changed. The particular changes depend on the type of index, as follows.

- If the index is a detail index, the procedure explained in Section b should be followed.
- If the index is an adaptive index, appropriate updates to INGRES tables SPLITINDG and SPLITINDX should be made. The discussion in Chapter II, Section E, indicates the nature of these changes.
- If the index is compound resource-based, then: 1) the individual resource categories should be entered into table INDXGENUS; 2) the auxiliary program GENMXVAL should be run; and 3) the auxiliary program GENIXRTYP should be run and table INDXRTYP should be changed, as described in Section C.4. (These changes are also applicable in the case of compound resource-based indices used as notional indices, as discussed in Section 1.c.5).

[14] This section discusses the use of new indices as added indices. Section 1.c.5) has discussed some issues regarding new indices that are to be used as notional indices.

Once the index is registered in the database, the procedure of Section c can be followed to specify the index as an added index for the appropriate variables. The caveats listed in Section c remain applicable. If a new adaptive index is invented, the restrictions stated in Chapter II, Section E, on the use of adaptive indices should be observed.

REFERENCES

[1] Anderson, Lowell Bruce, and Eleanor L. Schwartz, *NAVMOD: A Naval Model, Volume I: Main Report, and Volume II: Appendices*. IDA Report R-278, Institute for Defense Analyses, Alexandria, VA, October 1985.

[2] Schwartz, Eleanor L., *Modifications to NAVMOD*. IDA Paper P-2006, Institute for Defense Analyses, Alexandria, VA, June 1987.

[3] *Getting Started in INGRES*. Relational Technology, Inc., Alameda, CA, 1987.

[4] Date, C.J., *A Guide to INGRES*. Addison-Wesley, Reading, MA, 1987.

[5] *INGRES/SQL Self-Instruction Guide* (Release 5.0, VAX/VMS). Relational Technology, Inc., Alameda, CA, August 1986.

[6] *INGRES/SQL Reference Manual* (Release 5.0, VAX/VMS). Relational Technology, Inc., Alameda, CA, August 1986.

[7] *INGRES/QUEL Self-Instruction Guide* (Release 5.0, VAX/VMS). Relational Technology, Inc., Alameda, CA, August 1986.

[8] *INGRES/QUEL Reference Manual* (Release 5.0, VAX/VMS). Relational Technology, Inc., Alameda, CA, August 1986.

[9] *INGRES/EMBEDDED SQL User's Guide and Reference Manual* (Release 5.0, VAX/VMS). Relational Technology, Inc., Alameda, CA, August 1986.

[10] *INGRES/EMBEDDED SQL Companion Guide for FORTRAN* (Release 5.0, VAX/VMS). Relational Technology, Inc., Alameda, CA, August 1986.

[11] *INGRES/FORMS: Visual-Forms-Editor User's Guide* (Release 5.0, VAX/VMS). Relational Technology, Inc., Alameda, CA, August 1986.

[12] *INGRES/QUERY: Query-by-Forms User's Guide* (Release 5.0, VAX/VMS). Relational Technology, Inc., Alameda, CA, August 1986.

[13] *INGRES/MENU User's Guide* (Release 5.0, VAX/VMS). Relational Technology, Inc., Alameda, CA, August 1986.

[14] *INGRES Terminal User's Guide* (Release 5.0, VMS). Relational Technology, Inc., Alameda, CA, August 1986.

[15] Schwartz, Eleanor L., *Inputs to NAVMOD, With Definitions and Selected Characteristics, Arranged by Category*. Working Paper WP-87-1 of IDA Project 3661, Institute for Defense Analyses, Alexandria, VA, February 1987, revised October 1987.

[16] *How to Change Ownership of a Database*. INGRES Technical Note #20 (VMS Version), Relational Technology, Inc., Alameda, CA, December 1986.

APPENDIX A

TABULAR GUIDES TO THE AGGREGATOR PREPROCESSOR

CONTENTS OF APPENDIX A

TABLES

APPENDIX A

TABULAR GUIDES TO THE AGGREGATOR PREPROCESSOR

A. INTRODUCTION TO THE TABLES

This appendix consists of tables that provide certain information about the aggregator preprocessor.

Table A-1 gives a complete list of all the routines of the aggregator preprocessor computer program (main program, subroutines, and the one function subprogram), in alphabetical order by routine name, and indicates certain information about each routine. This information includes the INGRES tables each routine retrieves from and the INGRES tables it updates.

Table A-2 lists the routines of the aggregator preprocessor computer program in the order they appear in the main source code file. This order is about the same as the order the routines were discussed in Chapter III.

For each INGRES table in the database NAVPRE that is used by the aggregator, Table A-3 shows which routines retrieve from it and which routines (if any) update it. Note that Table A-3 (and Table A-1) applies only to the aggregator itself, not to the auxiliary programs. For information on the INGRES tables accessed by the auxiliary programs, see Chapter IV, Section C.

Table A-4 shows the logical units used by the aggregator preprocessor computer program. This table summarizes the information on input and output files that was discussed in Chapter I, Section C.

Table A-5 briefly describes the "included files" on which some parts of the source code are located. These included files are brought into the main source code file by means of INCLUDE statements, and some of them can be changed by auxiliary program CHGINP3 (see Chapter IV, Section C.1).

Table A-6 defines the major symbolic constants that the program uses. (In addition, the program uses some of the indices, most notably MXLOC, as symbolic constants; the included file NAVPARAM.PAR contains PARAMETER statements to define them.)

Tables A-1 through A-6 deal with the aggregator preprocessor as a computer program. Tables A-7 and A-8 focus on the database. Table A-7 shows the "detail" (index code 30) indices that currently are set up in the database, and the variables that use these indices (in the current version of the database). This table should be read in conjunction with Tables III-1 and III-2 of Chapter III. Of course, the user can change the added indices as desired. For definitions of the variables listed in Table A-7, see Appendix F of Reference [2].

Table A-8 lists the resource categories of the aggregator preprocessor and possible actual resource types that could be encompassed by these categories. The actual resource types shown are the ones currently set up in the database. Table A-8 was developed from the current version of INGRES table GENTYP0 by the auxiliary program GENREPORT, as described in Chapter IV, Section C.8.

Table A-9 provides a taxomony of the files associated with the aggregator preprocessor.

B. TABLES

Tables A-1 through A-9, below, conclude this appendix.

Table A-1. ROUTINES OF THE AGGREGATOR PREPROCESSOR

Subroutine AGGEFF(NAME, JV, IERR)

Mnemonic: "Aggregate effectiveness variables"
Purpose: Aggregates the more disaggregated data for a given effectiveness variable (NAME, JV) into data suitable for use by NAVMOD
Called by routines: EFFVAR
Calls/uses routines: PUTONE, GETALL, and EXTERNAL-declared subroutine WRITE2
COMMON blocks: CHRCMN
Included files: none
INGRES tables retrieved from: RVARINFO, INDXBINFO, GENUSTYWGT, INDXGENUS, GENUSINFO, INDXDTYP, INDXRTYP, SPLITINDG, SPLITINDX, LIMVINFO, RDATEFF
INGRES tables and columns updated: none

Program AGGPREPRO

Mnemonic: "Aggregator preprocessor"
Purpose: Main program of aggregator preprocessor; calls other routines as indicated by input options
Called by routines: none--this is the main program.
Calls/uses routines: POWSET, CLSET, GETIVA, CHRINI, STINDD, LMVSET, LMVCMP, RSOWGT, INDWGT, EFFVAR
COMMON blocks: CHRCMN, blank COMMON (also CBACK, CSTBF)
Included files: CMISUM.PAR, VALUS.CMN
INGRES tables retrieved from: PFNAMEF
INGRES tables and columns updated: none

Subroutine CHRINI

Mnemonic: "Character initialization"
Purpose: Encodes the integers 1 through 99 as character strings '01', '02', ..., '09', '10', '11',..., and stores the character strings in COMMON block CHRCMN for use by other routines
Called by routines: main program
Calls/uses routines: none
COMMON blocks: CHRCMN
Included files: none
INGRES tables retrieved from: none
INGRES tables and columns updated: none

(continued)

Table A-1 (continued)

Subroutine CLRWK

Mnemonic: "Clear working space"
Purpose: Resets to zero certain working arrays used by Subroutine RSOWGT
Called by routines: RSOWGT
Calls/uses routines: none
COMMON blocks: WKSPC
Included files: WKARY.CMN
INGRES tables retrieved from: none
INGRES tables and columns updated: none

Subroutine CLSET

Mnemonic: "Set COMMON lengths"
Purpose: Determine lengths of the sections in blank COMMON into which the NAVMOD inputs are partitioned; computes some working variables used by the preprocessor
Called by routines: main program
Calls/uses routines: none
COMMON blocks: IVARB, IVARBC
Included files: CMISUM.PAR, IVARB.CMN, CMILGTHS.TXT
INGRES tables retrieved from: none
INGRES tables and columns updated: none

Subroutine EFFVAR(IOPTN)

Mnemonic: "Effectiveness variables"
Purpose: Determines, based on input option choice, which effectiveness variables should be aggregated in this run of the preprocessor
Called by routines: main program
Calls/uses routines: AGGEFF, FINDVN
COMMON blocks: IVARB, IVARBC
Included files: CMISUM.PAR, IVARB.CMN
INGRES tables retrieved from: PFNAMEF, EVARINFO, EFFTABCUR
INGRES tables and columns updated: Table EFFTABCUR, column TABNAME

Subroutine FINDVN(NAME, NUMVAR)

Mnemonic: "Find variable number"
Purpose: Finds the number (subscript of array IVARQQ) of the NAVMOD input (scalar or array) having the name NAME
Called by routines: LMVSET, RSOWGT, EFFVAR
Calls/uses routines: none
COMMON blocks: IVARB, IVARBC, GETDB
Included files: CMISUM.PAR, IVARB.CMN, GETDB.CMN
INGRES tables retrieved from: none
INGRES tables and columns updated: none

(continued)

Table A-1. (continued)

Subroutine GETALL(JV, NAME, FUNC, VAL, ITP, MTA)
Mnemonic: "Get all values"
Purpose: Outputs the value of a NAVMOD input scalar variable, or the values of all elements of a NAVMOD input array, in a format readable by NAVMOD (these values are the aggregated data computed by the program)
Called by routines: LMVSET, RSOWGT, AGGEFF
Calls/uses routines: GETLD1
COMMON blocks: IVARB, IVARBC
Included files: CMISUM.PAR, IVARB.CMN
INGRES tables retrieved from: none
INGRES tables and columns updated: none

Subroutine GETIVA
Mnemonic: "Get IVARQ information"
Purpose: Reads in the "information file, which contains information on the names, characteristics, and storage of NAVMOD inputs. The information is stored in arrays IVARQQ and IVARQ (in COMMON blocks IVARQ (in COMMON blocks IVARBC and IVARB, respectively).
Called by routines: main program
Calls/uses routines: none
COMMON blocks: IVARB, IVARBC
Included files: CMISUM.PAR, IVARB.CMN
INGRES tables retrieved from: none
INGRES tables and columns updated: none

Subroutine GETLD1(JV, NAME, FUNC, ID, JD, KD, VAL, ITP, MTA)
Mnemonic: "Get one line of data"
Purpose: Outputs the value of a NAVMOD input scalar variable, or the values in one row, column, or "stab" of a NAVMOD input array, in a format readable by NAVMOD (these values are the aggregated data computed by the program)
Called by routines: GETALL
Calls/uses routines: Function ICEIL, Subroutine WRITE2, as passed into the argument "FUNC"
COMMON blocks: IVARB, IVARBC, COMIF1, blank COMMON (also CBACK, CSTBF)
Included files: CMISUM.PAR, VALUS.CMN, IVARB.CMN, COMIF1.CMN
INGRES tables retrieved from: none
INGRES tables and columns updated: none

(continued)

Table A-1. (continued)

Function ICEIL(X)

Mnemonic: "Ceiling over X"

Purpose: Computes the "ceiling function" $\lceil X \rceil$, i.e., the smallest integer greater than or equal to X (X is assumed to be nonnegative).

Called by routines: GETLD1

Calls/uses routines: none

COMMON blocks: none

Included files: none

INGRES tables retrieved from: none

INGRES tables and columns updated: none

Subroutine INDWGT

Mnemonic: "Indices and weights"

Purpose: Computes raw and normalized weights for the indices that comprise two or more resource categories (those indices with code 6 in table INDXBINFO)

Called by routines: main program

Calls/uses routines: none

COMMON blocks: WRTO (currently appears only in Subroutine INDWGT)

Included files: none

INGRES tables retrieved from: INDXBINFO, INDXGENUS, GENUSINFO, GENUSTYWGT, INDXRTYP

INGRES tables and columns updated: Table INDXRTYP, columns RAWWGT, RLTYPWGT

Subroutine LMVCMP

Mnemonic: "Computed limit variables"

Purpose: Computes appropriate numbers of elements of certain notional arrays to use (these correspond to computed limit variables in NAVMOD)

Called by routines: main program

Calls/uses routines: none

COMMON blocks: none

Included files: none

INGRES tables retrieved from: INDXBINFO, INDXGENUS, CMPLIM

INGRES tables and columns updated: Table INDXBINFO, columns IVALNTL, NSTEPS

(continued)

Table A-1. (continued)

Subroutine LMVSET

Mnemonic: "Set limit variables"

Purpose: Generates values for the limit variables, based on inputs found in the resources, ordnance, and limit variables file

Called by routines: main program

Calls/uses routines: FINDVN, PUTONE, GETALL, and EXTERNAL-declared subroutine WRITE2

COMMON blocks: none

Included files: NAVPARAM.PAR

INGRES tables retrieved from: LIMVINFO, RDATROL

INGRES tables and columns updated:

TABLE	COLUMN(S)
CMPLIM	STEPLVVAL
INDXBINFO	IVALNTL, NSTEPS
INDXGENUS	STEPLVVAL
GENUSINFO	NNTLTYP
LIMVINFO	LVVAL

Subroutine POWSET

Mnemonic: "Set powers"

Purpose: Initializes the array of powers of 2, which is used by the binary search performed in Subroutine FINDVN

Called by routines: main program

Calls/uses routines: none

COMMON blocks: GETDB

Included files: CMISUM.PAR, GETDB.CMN

INGRES tables retrieved from: none

INGRES tables and columns updated: none

Subroutine PUTONE(JV, ID, JD, KD, VAL)

Mnemonic: "Put one value"

Purpose: Stores the value (aggregation of more disaggregated data) of one NAVMOD input scalar or array element in the appropriate place in the blank COMMON block

Called by routines: LMVSET, RSOWGT, AGGEFF

Calls/uses routines: none

COMMON blocks: IVARB, IVARBC, blank COMMON (also CBACK, CSTBF)

Included files: CMISUM.PAR, VALUS.CMN, IVARB.CMN

INGRES files retrieved from: none

INGRES tables and columns updated: none

(continued)

Table A-1. (concluded)

Subroutine RSOWGT

Mnemonic: "Resources, ordnance, and weights"

Purpose: Computes notional values for the resource and ordnance values (these values can be used by NAVMOD) and computes some of the weights used for aggregation of effectiveness variables.

Called by routines: main program

Calls/uses routines: FINDVN, CLRWK, PUTONE, GETALL, and EXTERNAL-declared subroutine WRITE2

COMMON blocks: CHRCMN, WKSPC

Included files: WKARY.CMN

INGRES tables retrieved from: GENTYP0, LIMVINFO, RESGENUS, GENUSTYWGT, RDATROL

INGRES tables and columns updated: Table GENUSTYWGT, columns RAWWGT, RLTYPWGT

Subroutine STINDD

Mnemonic: "Set detail indices"

Purpose: Computes normalized weights for the "detail" indices (those with code 30 in table INDXBINFO) from the raw weights stored in table INDXDTYP

Called by routines: main program

Calls/uses routines: none

COMMON blocks: none

Included files: none

INGRES tables retrieved from: INDXBINFO, INDXDTYP

INGRES tables and columns updated: Table INDXBINFO, column IVALRL
Table INDXDTYP, column RLTYPWGT

Subroutine WRITE2(EDBUFF, IFMT, IST, IMAX, INC, NFN, VAL)

Mnemonic: "Writing subroutine (revision number) 2"

Purpose: (Internally) writes into the character string EDBUFF values of certain NAVMOD inputs (aggregated data), stored in the blank COMMON block

Called by routines: GETLD1 (via the passed argument FUNC)

Calls/uses routines: none

COMMON blocks: blank COMMON (also CBACK, CSTBF)

Included files: CMISUM.PAR, VALUS.CMN

INGRES tables retrieved from: none

INGRES tables and columns updated: none

Table A-2. ROUTINES OF THE AGGREGATOR PREPROCESSOR IN ORDER OF APPEARANCE IN THE COMPUTER CODE

1. Program AGGPREPRO (main program)
2. Subroutine STINDD
3. Subroutine LMVSET
4. Subroutine LMVCMP
5. Subroutine RSOWGT
6. Subroutine CLRWK
7. Subroutine INDWGT
8. Subroutine EFFVAR
9. Subroutine AGGEFF
10. Subroutine PUTONE
11. Subroutine POWSET
12. Subroutine GETIVA
13. Subroutine CLSET
14. Subroutine WRITE2
15. Subroutine GETALL
16. Subroutine GETLD1
17. Function ICEIL
18. Subroutine FINDVN
19. Subroutine CHRINI

Table A-3. INGRES TABLES IN DATABASE NAVPRE THAT ARE ACCESSED BY THE AGGREGATOR PREPROCESSOR[a]

Table Name	Subroutines that Retrieve from Table	Subroutines that Update Table, and Column(s) Updated[b]
CMPLIM	LMVCMP	LMVSET (column STEPLVVAL)
EFFTABCUR	EFFVAR	EFFVAR (column TABNAME)
EVARINFO	EFFVAR	None
GENTYP0	RSOWGT	None
GENUSINFO	AGGEFF, INDWGT	LMVSET (column NNTLTYP)
GENUSTYWGT	AGGEFF, INDWGT, RSOWGT	RSOWGT (columns RAWWGT, RLTYPWGT)
INDXBINFO	AGGEFF, INDWGT, LMVCMP, STINDD	LMVCMP (columns IVALNTL, NSTEPS) LMVSET (columns IVALNTL, NSTEPS) STINDD (column IVALRL)
INDXDTYP	AGGEFF, STINDD	STINDD (column RLTYPWGT)
INDXGENUS	AGGEFF, INDWGT, LMVCMP	LMVSET (column STEPLVVAL)
INDXRTYP	AGGEFF, INDWGT	INDWGT (columns RAWWGT, RLTYPWGT)
LIMVINFO	AGGEFF, LMVSET, RSOWGT	LMVSET (column LVVAL)
PFNAMEF	main program, EFFVAR	None
RDATEFF	AGGEFF	EFFVAR (whole table)
RDATROL	LMVSET, RSOWGT	None
RESGENUS	RSOWGT	None
RVARINFO	AGGEFF	None
SPLITINDG	AGGEFF	None
SPLITINDX	AGGEFF	None

[a]The tables INDXMSG and VARDEFS are not accessed by the aggregator preprocessor itself, but the auxiliary program DIMREPORT retrieves information from them. (Table VARDEFS is also used in the interaction selector preprocessor.)

[b]Some columns of some tables are not updated by the aggregator preprocesor itself, but can be updated by some of the auxiliary programs.

Table A-4. AGGREGATOR PREPROCESSOR LOGICAL UNITS

Logical Unit	Type	Contents
3	Input	File of option codes, indicating the portions of the preprocessor to be executed in the current run ("input options guide file")
4	Output	File of informative messages
7	Output	File of timings
17	Output	Main output of preprocessor -- output file of aggregated data. Given the name (via an OPEN statement) RESORDLM.AGD (for the resources,ordnance and limits portion of the preprocessor) or EFFVARS.AGD (for the effectiveness variables portion).
22	Input	"Information file" containing information on the names, dimensioning, storage locations, et al., of the NAVMOD inputs

Table A-5. AGGREGATOR PREPROCESSOR--BRIEF DESCRIPTIONS OF INCLUDED FILES

Included File Name	Used in Routines	Brief Description of Contents
CMILGTHS.TXT	CLSET	Portion of code for Subroutine CLSET; gives lengths of sections of blank COMMON block
CMISUM.PAR	(many)	PARAMETER statements to declare sizes of several frequently-used arrays
COMIF1.CMN	GETLD1	Declaration of COMMON block COMIF1, which contains format specifications for output file[a]
GETDB.CMN	FINDVN, POWSET	Declaration of COMMON block GETDB, which contains the powers of 2, used by the binary search routine in Subroutine FINDVN
IVARB.CMN	(many)	Declarations of COMMON blocks IVARB and IVARBC, which contain information about the names, dimensioning, and storage locations of the inputs to NAVMOD
NAVPARAM.PAR	LMVSET	PARAMETER statements for symbolic constants used in NAVMOD. This file is the same as the included file NAVPARAM.PAR used in NAVMOD.
VALUS.CMN	main program, PUTONE, WRITE2, GETLD1	Specification statements for the blank COMMON block and the COMMON blocks CBACK and CSTBF (the arrays in these COMMON blocks contain certain values of NAVMOD inputs).
WKARY.CMN	RSOWGT, CLRWK	Declaration of the COMMON block WKSPC, which contains several arrays that hold certain intermediate quantities computed by Subroutine RSOWGT. Also contains PARAMETER statements that define the sizes of these arrays.

[a]Subroutine GETLD1 is also used in the interaction selector preprocessor, where the relevant information is located in a COMMON block because it is also used by other routines. In the aggregator preprocessor, this is not necessary, but it was desired to have Subroutine GETLD1 be identical in both preprocessors.

Table A-6. AGGREGATOR PREPROCESSOR: DEFINITIONS OF SELECTED SYMBOLIC CONSTANTS[a]

Name of Constant	In Included File	Current Value[b]	Definition
MAXNV	CMISUM.PAR	800	An upper bound on the number of inputs to NAVMOD.
NUMCB	CMISUM.PAR	19	Number of sections into which the NAVMOD inputs are organized.
ICMX	CMISUM.PAR	6439	Number of elements in the largest section of NAVMOD inputs.
ICSUM	CMISUM.PAR	19223	Total number of elements occupied by the NAVMOD inputs.
MXVSIZ	CMISUM.PAR	2184	Number of elements in the NAVMOD input array that has the most elements.
LVEC	WKARY.CMN	40	Size of working array WKVEC[c]
L1	WKARY.CMN	40	Size of first dimension of working array WKARY1[c]
L2	WKARY.CMN	40	Size of second dimension of working array WKARY1[c]
L3	WKARY.CMN	40	Size of first dimension of working array WKARY2[c]
L4	WKARY.CMN	40	Size of second dimension of working array WKARY2[c]
MQNODN	d	3	Upper bound on number of notional dimensions for a NAVMOD input
MQNODT	d	6	Upper bound on total number of dimensions (notional plus added) for a NAVMOD input
MQR1	d	70	Upper bound on number of actual weapon types encompassed by any one index
MQSTP	d	20	Upper bound on number of steps in a compound index (except for numerical indices)
MQT1	d	11	Upper bound on number of steps in a compound index (except for numerical indices)
MQT2	d	20	Same as MQSTP
MQW1	d	20	Upper bound on number of notional weapon types associated with a non-numerical index
MQW2	d	5	Not currently used
MXRLTY	e	40	Upper bound on number of actual weapon types in any one resource category

Notes on next page (continued)

Table A-6. (concluded)

[a]Some of the names of the indices, in particular, MXLOC, the upper limit on the number of regions, are also used as symbolic constants in the aggregator.

[b]These current values might need to be changed if the NAVMOD inputs are changed or if more actual weapon types are considered.

[c]These symbolic constants should all have a value strictly greater than all of: 1) the number of regions played; 2) the number of actual weapon types in any resource category, and 3) the number of notional types for any resource.

[d]Not located in an included file, but used in Subroutine AGGEFF.

[e]Not located in an included file, but used in Subroutine RSOWGT.

Table A-7. POSSIBLE DETAIL INDICES AND VARIABLES THAT MIGHT USE THEM

JABCML - Blue method of launching land-attack cruise missiles
Might be used for variables that concern the launch of such missiles: ACCLCT, ACCLSL, PPCLCT, PPCLSL

JABDSM - Blue submarine screen protocol
(not curently specified as an added index for any variable, but might be useful as an added index for variables that involve Blue barrier submarines)

JABHEL - Blue sea-based ASW helicopter type
Might be used for variables used in the modeling of reactive helicopter ASW (Subroutine CTFMOD): COBASR, CUBASR, CXOASR, CXUASR, PKHBSS, ZLAMPF

JARHEL - Red sea-based ASW helicopter type
Might be used for variables used in the modeling of Red helicopter ASW in Subroutine MOVRS: RACCDW, RACPCD, RACPCK, RSORA

JARSBA - Red sea-based (fighter) aircraft
Might be used for variables used in the modeling of Red sea-based defensive aircraft (Subroutines ELSLBA and SHPSHP): EAAKAD, EAAKDA, EAAKED, EAAMPD, EAAWFF, EAORDA, EAVRA, RAPRS, SAAKAD, SAAKDA, SAAKED, SAAMPD, SAAWFF, SAORDA, SAVRA

Table A-8. Resource Categories and Possible Actual Resource Types

```
Category BAAEW :  Blue sea-based AEW aircraft
    Number of notional types is  1
    Potential actual types are:
        1  E-2

Category BAASW :  Blue sea-based ASW aircraft
    Number of notional types is  1
    Potential actual types are:
        1  S-3
        2  SH-3
        3  SH-2
        4  SH-60

Category BAATT :  Blue sea-based attack aircraft
    Number of notional types is  1
    Potential actual types are:
        1  A-6
        2  A-7
        3  F/A-18
        4  S-3

Category BAFTR :  Blue sea-based fighter aircraft
    Number of notional types is  1
    Potential actual types are:
        1  F-14
        2  F/A-18

Category BFBT   :  Blue surface ships—barrier tenders
    Number of notional types is  1
    Potential actual types are:
        1  NTL.BLU BT

Category BFCAR :  Blue surface ships—carriers
    Number of notional types is  1
    Potential actual types are:
        1  CVN-68
        2  CV-67
        3  CVN-65
        4  CV-63
        5  CV-59
        6  CV-41

Category BFESC :  Blue surface ships—escort ships
    Number of notional types is given by the limit variable NKBES
    Potential actual types are:
        1  CG-47
        2  CG-52
        3  CG-26
        4  CG-16
        5  CGN-38
        6  CGN-35
        7  CGN-36
        8  CGN-25
        9  CGN-9
       10  BB-61
       11  DDG-993
       12  DDG-51
       13  DDG-37
       14  DDG-2
       15  DD-963
       16  DD-963 ABL
       17  DD-963 VLS
       18  FFG-7
       19  FFG-1
       20  FF-1052
       21  FF-1040
```


Continued

Table A-8. (continued)

```
Category BFURG :  Blue surface ships—URG ships
    Number of notional types is  1
    Potential actual types are:
        1  AOE
        2  AOR
        3  AO
        4  AE
        5  AFS

Category BLAEW :  Blue land-based AEW aircraft
    Number of notional types is  1
    Potential actual types are:
        1  E-3

Category BLASW :  Blue land-based ASW/ASuW aircraft
    Number of notional types is  1
    Potential actual types are:
        1  P-3

Category BLFTR :  Blue land-based fighter aircraft
    Number of notional types is given by the limit variable NKBDPL
    Potential actual types are:
        1  F-15
        2  F-16
        3  B-52

Category BOAAA :  Blue air-to-air munitions for sea-based aircraft
    Number of notional types is  1
    Potential actual types are:
        1  AIM-7
        2  AIM-9
        3  AIM-54

Category BOAAF :  Blue anti-surface munitions for sea-based aircraft
    Number of notional types is  1
    Potential actual types are:
        1  HARPOON

Category BOAAG :  Blue air-to-ground munitions for sea-based aircraft
    Number of notional types is  1
    Potential actual types are:
        1  G/P BOMB
        2  LS/GD BOMB
        3  ROCKEYE
        4  WALLEYE
        5  MAVERICK
        6  SHRIKE
        7  HARM

Category BOAAS :  Blue ASW munitions for sea-based aircraft
    Number of notional types is  1
    Potential actual types are:
        1  MK-46
        2  MK-50

Category BOBAR :  Blue munitions for barrier submarines
    Number of notional types is  1
    Potential actual types are:
        1  MK-37
        2  MK-48
        3  ADCAP

Category BOFCM :  Blue land-attack cruise missiles for surface ships
    Number of notional types is  1
    Potential actual types are:
        1  TLAM
        2  TLAM/C
        3  TLAM/D
```


Continued

Table A-8. (continued)

Category BOFDA : Blue AAD munitions for surface ships
Number of notional types is 1
Potential actual types are:
1 SM-1-ER
2 SM-2-ER
3 SM-1-MR
4 SM-2-MR

Category BOFDS : Blue SSD munitions for surface ships
Number of notional types is 1
Potential actual types are:
1 NATO SS

Category BOFFF : Blue anti-surface munitions for surface ships
Number of notional types is 1
Potential actual types are:
1 HARPOON
2 TOMAHAWK

Category BOFTP : Blue ASW munitions for surface ships
Number of notional types is 1
Potential actual types are:
1 MK-46
2 MK-50

Category BOLAA : Blue air-to-air munitions for land-based aircraft
Number of notional types is 1
Potential actual types are:
1 AMRAAM
2 SPARROW
3 SIDEWINDER

Category BOLAF : Blue anti-surface munitions for land-based aircraft
Number of notional types is 1
Potential actual types are:
1 HARPOON

Category BOLAS : Blue ASW munitions for land-based aircraft
Number of notional types is 1
Potential actual types are:
1 MK-46
2 MK-50

Category BOSCM : Blue land-attack cruise missiles for dir.-supt. subs
Number of notional types is 1
Potential actual types are:
1 TLAM
2 TLAM/C
3 TLAM/D

Category BOSTP : Blue ASW/ASuW munitions for direct-support subs
Number of notional types is 1
Potential actual types are:
1 MK-48
2 ADCAP
3 HARPOON
4 TASM

Category BSBAR : Blue barrier submarines
Number of notional types is 1
Potential actual types are:
1 SSN-688
2 SSN-688VLS
3 SSN-637
4 SSN-594
5 SSN-585

Continued

Table A-8. (continued)

```
Category BSDS  :  Blue direct-support submarines
    Number of notional types is  1
    Potential actual types are:
        1  SSN-688
        2  SSN-637
        3  SSN-594
        4  SSN-585
        5  SSN-688VLS

Category BUA   :  Blue sonobuoys for sea-based patrol ASW aircraft
    Number of notional types is  1
    Potential actual types are:
        1  SSQ-41
        2  SSQ-53
        3  SSQ-62
        4  SSQ-AA
        5  SSQ-BB

Category BUL   :  Blue sonobuoys for land-based patrol ASW aircraft
    Number of notional types is  1
    Potential actual types are:
        1  SSQ-41
        2  SSQ-53
        3  SSQ-62
        4  SSQ-AA
        5  SSQ-BB

Category BUR   :  Blue sonobuoys for reactive (sea-based) ASW helos
    Number of notional types is  1
    Potential actual types are:
        1  SSQ-41
        2  SSQ-47
        3  SSQ-CC
        4  SSQ-DD

Category RFBT  :  Red  surface ships—barrier tenders
    Number of notional types is  1
    Potential actual types are:
        1  NTL.RED BT

Category RFNRP :  Red  surface ships—non-resupply
    Number of notional types is given by the limit variable NKRSN
    Potential actual types are:
        1  KIEV CVHG
        2  KIROV CGN
        3  MOSKVA CHG
        4  KARA CG
        5  KYNDA CG
        6  SLAVA CG
        7  KRESTA 1
        8  KRESTA 2
        9  UDALOY DDG
       10  SOVREMENNY
       11  KASHIN DDG
       12  KANIN DDG
       13  KOTLIN DDG
       14  MOD KOTLIN
       15  KRIVAK 1
       16  KRIVAK 2
       17  KRIVAK 3
       18  MOD KASHIN
       19  SKORYY/MOD
       20  MOD KILDIN
       21  SVERDLOV
       22  NANUCH 1,3
```


Continued

Table A-8. (continued)

```
        23  NANUCH 2
        24  PETYA1,2,3
        25  MOD PETYA
        26  RIGA FF
        27  MIRKA 1,2
        28  GRISHA 134
        29  PARCHIM 2
        30  TARANTUL

Category RFRSP :  Red  surface ships—resupply ships
    Number of notional types is  1
    Potential actual types are:
         1  NTL.RED RS

Category RLBMR :  Red  land-based aircraft—bomber
    Number of notional types is given by the limit variable NKRB
    Potential actual types are:
         1  BACKFIRE
         2  BADGER C/G
         3  BLINDER
         4  BEAR B/C/G
         5  BLACKJACK
         6  FENCER
         7  FITTER

Category RLFTR :  Red  land-based aircraft—fighter/escort
    Number of notional types is  1
    Potential actual types are:
         1  FENCER
         2  FITTER
         3  FLAGON
         4  FLOGGER
         5  FOXBAT
         6  FOXHOUND
         7  FLANKER
         8  FULCRUM
         9  FIDDLER
        10  FISHBED

Category RLINT :  Red  land-based aircraft—interceptor
    Number of notional types is  1
    Potential actual types are:
         1  FENCER
         2  FITTER
         3  FLAGON
         4  FLOGGER
         5  FOXBAT
         6  FOXHOUND
         7  FLANKER
         8  FULCRUM
         9  FIDDLER
        10  FISHBED

Category RMABD :  Red  land-based SAMs for airbase defense
    Number of notional types is given by the limit variable NABSAM
    Potential actual types are:
         1  SA-2
         2  SA-3
         3  SA-5

Category RMPPD :  Red  land-based SAMs for power-projection defense
    Number of notional types is given by the limit variable NPPSAM
    Potential actual types are:
         1  SA-2
         2  SA-3
         3  SA-5
```


Continued

Table A-8. (continued)

Category ROBAR : Red munitions for barrier submarines
Number of notional types is 1
Potential actual types are:
1 53-83
2 65-76
3 E65-83
4 SET-65 M
5 53 S

Category ROFAA : Red air-to-air munitions for sea-based aircraft
Number of notional types is 1
Potential actual types are:
1 ATOLL
2 APHID
3 KERRY

Category ROFAS : Red air-dropped ASW munitions for surface ships
Number of notional types is 1
Potential actual types are:
1 PLAB
2 E45
3 E53
4 53 S

Category ROFDA : Red AAD munitions for surface ships
Number of notional types is 1
Potential actual types are:
1 SA-N-3
2 SA-N-6
3 SA-N-7

Category ROFDL : Red long-range SAMs for surface ships
Number of notional types is 1
Potential actual types are:
1 SA-N-3
2 SA-N-6
3 SA-N-7

Category ROFDS : Red SSD munitions for surface ships
Number of notional types is 1
Potential actual types are:
1 SA-N-1
2 SA-N-3
3 SA-N-4
4 SA-N-6
5 SA-N-7
6 SA-N-5
7 SA-NX-9

Category ROFFL : Red long-range ASuW munitions for surface ships
Number of notional types is 1
Potential actual types are:
1 SS-N-2c
2 SS-N-3b
3 SS-N-9
4 SS-N-12
5 SS-N-19
6 SS-N-22

Category ROFFR : Red regular-range ASuW munitions for surface ships
Number of notional types is 1
Potential actual types are:
1 SS-N-2c
2 SS-N-3b
3 SS-N-9
4 SS-N-12
5 SS-N-14
6 SS-N-19
7 SS-N-22

Continued

Table A-8. (continued)

```
Category ROFTP :  Red  ASW munitions for surface ships
    Number of notional types is  1
    Potential actual types are:
        1  53
        2  53 S
        3  SET-65

Category ROLAA :  Red  air-to-air munitions for land-based aircraft
    Number of notional types is  1
    Potential actual types are:
        1  AA-2
        2  AA-3
        3  AA-5
        4  AA-6
        5  AA-7
        6  AA-8
        7  AA-9
        8  AA-10
        9  AA-11

Category ROLAF :  Red  anti-surface munitions for land-based aircraft
    Number of notional types is  1
    Potential actual types are:
        1  AS-4
        2  AS-5
        3  AS-6
        4  AS-2
        5  AS-3

Category ROSCM :  Red  attack munitions for missile-firing submarines
    Number of notional types is  1
    Potential actual types are:
        1  SS-N-3a
        2  SS-N-7
        3  SS-N-9
        4  SS-N-12
        5  SS-N-19

Category ROSTP :  Red  attack munitions for torpedo-firing submarines
    Number of notional types is  1
    Potential actual types are·
        1  65-76
        2  E65-83
        3  53-83
        4  SET-65 M
        5  53 S

Category RSBAR :  Red  barrier submarines
    Number of notional types is  1
    Potential actual types are:
        1  ALFA
        2  VICTOR I
        3  NOVEMBER
        4  TANGO
        5  FOXTROT
        6  AKULA
        7  SIERRA
        8  MIKE
        9  YANKEE
       10  VICTOR II
       11  VICTOR III
       12  ECHO
       13  WHISKEY
       14  KILO
```


Continued

Table A-8. (concluded)

```
Category RSCM  :  Red  missile-firing submarines
    Number of notional (sub)types is set to 4, each subtype
    having its own characteristics.
    Potential actual types are:
        1  OSCAR
        2  CHARLIE I
        3  CHARLIE II
        4  PAPA
        5  ECHO II
        6  JULIETT
        7  MOD ECHOII

Category RSCMA :  Red  missile-firing submarines, aggregated to 1 type
    Number of notional types is  1
    Potential actual types are:
        1  OSCAR
        2  CHARLIE I
        3  CHARLIE II
        4  PAPA
        5  ECHO II
        6  JULIETT
        7  MOD ECHOII

Category RSTP  :  Red  torpedo-firing submarines
    Number of notional types is  1
    Potential actual types are:
        1  ALFA
        2  VICTOR I
        3  NOVEMBER
        4  TANGO
        5  FOXTROT
        6  AKULA
        7  SIERRA
        8  MIKE
        9  YANKEE
       10  VICTOR II
       11  VICTOR III
       12  ECHO
       13  WHISKEY
       14  KILO
```

Table A-9. TAXONOMY OF AGGREGATOR PREPROCESSOR FILES

PROGRAM FILES

Program Source Code Files

Main embedded SQL source code file AGGPREPRO.SF

Included files--see Table A-5

Produced Program Files

FORTRAN file AGGPREPRO.FOR--output by embedded SQL preprocessor

Object file AGGPREPRO.OBJ--output by FORTRAN compiler

Executable file AGGPREPRO.EXE--output by linker, executable file of the preprocessor program

INPUT FILES

Input options guide file (whatever name the user desires; user should associate file with logical unit 3)

Information file BLKINP.DAT (user should associate file with logical unit 22)

File of more disaggregated data for resource, ordnance, limit, and allocation fraction variables (user specifies name of file)

Files of more disaggregated data for effectiveness variables (must be consistent with information in INGRES database table EVARINFO and PFNAMEF)

OUTPUT FILES

RESORDLM.AGD--file of data lines (readable by NAVMOD) containing aggregated values of the resource, ordnance, and limit variables. User can rename this file as desired.

EFFVARS.AGD--file of data lines (readable by NAVMOD) containing aggregated values of the effectiveness variables. User can rename this file as desired.

File of informative messages (associated with logical unit 4)

File of timings (associated with logical unit 7)

INGRES DATABASE FILES

An INGRES database can be transferred from one machine to another via a series of INGRES-supplied procedures. First, an unload procedure produces a set of files that

(continued)

Table A-9. (concluded)

contain the database information in encoded form. There is one file for every table in the database, plus a number of additional files. These files can be transferred to another machine that has INGRES installed on it. Then, on the second machine, a reload procedure can construct from the files a database that is identical (from the user's point of view) to the original database. (See References [6] and [16] for more information.). These files can be considered "files associated with the aggregator preprocessor," with the understanding that after the database has been set up on the desired machine, the files themselves are not directly relevant to the preprocessor.

FILES FOR AUXILIARY PROGRAMS

Program Source Code Files

Main source code files--see Chapter IV, Section C.

Included files. The information file BLKINP.DAT is included in auxiliary program DIMREPORT. Some auxiliary programs have a few other included files, which are also included files for the aggregator itself, and appear in Table A-5.

Produced Program Files

FOR¯RAN, object, and executable files that result from processing the source code files.

File DCOMMN.DAT of symbolic constants, COMMON block names, variable names, and dimensioning of variables. Read by CHGINP3.

Output Files--see Chapter IV, Section C.

APPENDIX B

DEFINITIONS OF TABLES AND COLUMNS IN THE DATABASE NAVPRE

APPENDIX B

DEFINITIONS OF TABLES AND COLUMNS IN THE DATABASE NAVPRE

This appendix provides brief definitions of the tables in the database NAVPRE that are used by the aggregator and/or its auxiliary programs. All of the columns of these tables are also defined.

Certain other tables in the database NAVPRE are used by the interaction selector preprocessor. Only table EVARINFO is used by both preprocessors. Table VARDEFS is used by the interaction selector preprocessor and by the auxiliary program DIMREPORT (see Chapter IV, Section C.9), but not by the aggregator itself.

INGRES DATABASE NAVPRE—TABLES USED IN THE AGGREGATOR PREPROCESSOR AND/OR ITS AUXILIARY PROGRAMS

Table: CMPLIM ("computed limits")
Row width: 23
Description: information on the components of certain numerical compound indices
Columns and their meanings:

Column Name	Type	Length	Meaning
indxnam	character	6	name of index
istep	integer	1	"step" number (1, 2, 3,...)
stepindx	character	6	name of index corresponding to this step
stepixval	integer	2	maximum value (dimension limit) for index stepindx
steplvnam	character	6	limit variable (if any) corresponding to the index for this step; if no such limit variable, stepindx itself is used
steplvval	integer	2	value of the limit variable in preceding column, or stepixval if no such limit variable

Table: EFFTABCUR ("effectiveness table current")
Row width: 45 (there must be exactly one row)
Description: Name of flat file the contents of which are currently in the table rdateff (file name and extension only; directory specification is in table pfnamef).
Columns and their meanings:

Column Name	Type	Length	Meaning
tabname	character	45	(see definition above)

Table: EVARINFO ("extra variable information")
Row width: 76
Description: information on the input variables used by NAVMOD (encompassing effectiveness variables, resource variables, ordnance stock variables and input limit variables) Most of this information is similar to that of Appendix F of IDA Paper P-2006, q.v. This table is used by both the aggregator and the interaction selector preprocessors.
Columns and their meanings:

Column Name	Type	Length	Meaning
vname	character	6	name of variable
increm	integer	1	0 if the variable value can be replaced by Subroutine TIMET; 1 if the variable value can be incremented by Subroutine TIMET;
cblock	character	6	name of COMMON block in which the variable is located in NAVMOD
rstcode	integer	1	restriction code—see table rstmsg
rout	character	6	name of subroutine of NAVMOD in which the variable is used; 'MULTPL' for variables used in more than one subroutine
catname	character	6	name of category to which the variable is assigned. See Appendix F of IDA Paper P-2006 for information about these categories. Category specification consists of the category name plus category number. Category name is often the same as the subroutine name in the "rout" column; 'MULTPL' indicates variables belonging to more than one category; 'GENRAL' indicates variables belonging to a "general category" (continued)

Table EVARINFO, continued

Column Name	Type	Length	Meaning
catno	integer	1	number of category to which the variable is assigned.
ndatlin	integer	4	number of data lines for this variable in the file indicated in the "filname" column
filname	character	45	name of flat file containing the more disaggregated data for this variable (file name and extension only; directory specification is in table pfnamef). This file is read into table rdateff, whence its data are read by the aggregator.

Table: GENTYP0 ("genus types initial")
Row width: 18
Description: Actual weapon types associated with each resource category (The resource categories are fixed, in accordance with the categories of resources that NAVMOD can simulate. The weapon types can be reset by the user.)
Columns and their meanings:

Column Name	Type	Length	Meaning
genusnam	character	6	code name of resource category
ordno	integer	2	order number of actual weapon type within this resource category
rltyp	character	10	10-character code name of weapon type

Table: GENUSINFO ("genus information")
Row width: 68
Description: information on the resource categories used by the aggregator preporcessor
Columns and their meanings:

Column Name	Type	Length	Meaning
genusnam	character	6	code name of the resource category
genuscode	integer	1	code number of resource category, as follows: 1—NAVMOD simulates one notional type of this resource 2—number of types of this resource that are simulated is given by a limit variable 3—NAVMOD simulates fixed number of types of this resource (not currently applicable) 4—NAVMOD simulates fixed number of subtypes of this resource; each subtype can have differences in its modeling 5—resource category corresponds to an aggregation of subtypes in some code-4 catgegory
nrltyp	integer	2	number of actual weapon types in this resource category (see table gentyp0)
nntltyp	integer	2	number of notional types of this resource that NAVMOD simulates
ldnam	character	6	name of limit variable (for code-1 resource categories) or name of index ,if any, (for code-3 or code-4 resource categories) associated with this resource category
side	character	4	"Blue" or "Red"
genusmemo	character	47	brief description of this category

Table: GENUSTYWGT ("genus types and weights")
Row width: 28
Description: For each resource category, relative weights for the actual weapon types for each notional type NAVMOD will simulate.

Columns and their meanings:

Column Name	Type	Length	Meaning
genusnam	character	6	code name of resource category
ntlindval	integer	2	notional resource type played (ranges from 1 to the number of notional types of this resource that NAVMOD simulates [see table genusinfo]). A value of zero indicates a total over all notional resource types.
ordno	integer	2	order number of actual weapon type (see table gentyp0)
rltyp	character	10	code name of actual weapon type (see table gentyp0)
rawwgt	float	4	raw weight for *indicated actual weapon* type that is considered to be notional type ntlindval (computed by program based on sum of appropriate resource variables)
rltypwgt	float	4	normalized weight for indicated actual weapon type that is considered to be notional type ntlindval (computed by program from column rawwgt)

Table: INDXBINFO ("index basic information")
Row width: 26
Description: Basic information about the indices. These indices are used as labels pointing to groups of actual weapon types and numbers of notional types; some of them are used as symbolic constants in the NAVMOD computer code. See IDA Paper P-2006, Chapter III.

Columns and their meanings:

Column Name	Type	Length	Meaning
indxnam	character	6	name of index
indxcode	integer	1	code number of index. see documentation for explanation of meaning of different code numbers
ivalrl	integer	2	number of actual weapon types associated with this index
mxvalntl	integer	2	upper limit on the numerical value associated with this index in the NAVMOD program itself (value of symbolic constant/dimension limit)
ivalntl	integer	2	actual numerical value associated with this index in the NAVMOD program itself (value of associated limit variable; number of notional types simulated of resources associated with this index)
nsteps	integer	1	number of distinct resource categories or "component" indices associated with this index
genus	character	6	code name of associated resource category (if there is exactly one such)
indxrnam	character	6	name of "related" index (used if indxcode=40)

Table: INDXDTYP ("'detail' index types")
Row width: 26
Description: actual weapon types and relative weights for indices that encompass more detail than NAVMOD simulates
Columns and their meanings:

Column Name	Type	Length	Meaning
indxnam	character	6	name of index
ordno	integer	2	order number of actual weapon type
rltyp	character	10	10-character code name of actual weapon type
rawwgt	float	4	relative weight for this actual weapon type (must be supplied by user)
rltypwgt	float	4	normalized weight for this actual weapon type (computed by program from preceding column)

Table: INDXGENUS ("indices and genuses")
Row width: 15
Description: for indices which encompass more than one resource category, the resource categories, in order, associated with that index
Columns and their meanings:

Column Name	Type	Length	Meaning
indxnam	character	6	name of index
istep	integer	1	step number (1, 2, ...)
stepgenus	character	6	code name of resource category associated with step istep of index indxnam
steplvval	integer	2	number of notional types associated with the above resource category (see table genusinfo)

Table: INDXGTYP ("indices, genuses, and types")
this table is a view, combining data from the tables indxgenus and gentyp0
Row width: 25
Description: actual weapons types associated with indices that encompass more than one resource category
Columns and their meanings:

Column Name	Type	Length	Meaning
indxnam	character	6	name of index
istep	integer	1	step number (of resource category within index; see table indxgenus)
genusnam	character	6	code name of resource category for above step
ordno	integer	2	order number of actual weapon type in this resource category
rltyp	character	10	name of actual weapon type in this resource category

Table: INDXMSG ("index message")
Row width: 66
Description: Mnemonic message describing each index. Not used by aggregator itself, but is used in auxiliary programs.
Columns and their meanings:

Column Name	Type	Length	Meaning
indxnam	character	6	name of index
msg	character	60	associated descriptive message

Table: INDXRTYP ("indices, real types, and weights")
Row width: 26
Description: Actual weapon types and weights associated with indices that have more than one associated resource category. The set of weapon types is the union of the sets of weapon types associated with each resource category associated with this index (see table indxgenus).
Columns and their meanings:

Column Name	Type	Length	Meaning
indxnam	character	6	name of index
ordno	integer	2	order number of actual weapon type
rltyp	character	10	10-character code name of actual weapon type
rawwgt	float	4	relative weight for this actual weapon type (computed by program)
rltypwgt	float	4	normalized weight for this actual weapon type (computed by program)

Table: LIMVINFO ("limit variable information")
Row width: 32
Description: information on limit variables and their values; information on numbers of types of resources simulated
Columns and their meanings:

Column Name	Type	Length	Meaning
lvnam	character	6	name of limit variable (can be blank for code 3 and code 4 resource categories, which should be listed in this table)
lvval	integer	4	value of limit variable (if lvnam is blank, then number of types of resource simulated; should agree with table genusinfo)
dimnam	character	6	name of symbolic constant associated with limit variable or resource category
dimval	integer	4	value of such symbolic constant
genus	character	6	code name of resource category, if any, associated with limit variable
allocvar	character	6	name of "allocation variable" governing assignment of actual types in the resource category to notional types. Blank if no associated resource category.

Table: *PFNAMEF ("prefix for full file name")*
Row width: 45 (there must be exactly one row)
Description: Device and directory specification of flat data files that contain the input data values for the aggregator. All such files are assumed to reside in this directory. This table should contain only one row.
Columns and their meanings:

Column Name	Type	Length	Meaning
pfname	character	45	(see above)

Table: RDATEFF ("real data—effectiveness ")
Row width: 71
Description: More disaggregated data for effectiveness variables. Aggregated by aggregator into effectiveness inputs for NAVMOD.
Columns and their meanings:

Column Name	Type	Length	Meaning
vnamer	character	6	name of input variable
fcati	integer	1	(not currently used)
comp1	character	10	(code name of) actual weapon type in component 1 (if any)
comp2	character	10	(code name of) actual weapon type in component 2 (if any)
comp3	character	10	(code name of) actual weapon type in component 3 (if any)
comp4	character	10	(code name of) actual weapon type in component 4 (if any)
comp5	character	10	(code name of) actual weapon type in component 5 (if any)
comp6	character	10	(code name of) actual weapon type in component 6 (if any)
val	float	4	data value

Table: RDATROL ("real data—resources, ordnance, and limits")
Row width: 71
Description: more disaggregated data values for resources and ordnance stocks, and desired values for limit variables and allocation variables
Columns and their meanings:

Column Name	Type	Length	Meaning
vnamer	character	6	name of input variable
fcati	integer	1	(not currently used)
comp1	character	10	(code name of) actual weapon type in component 1 (if any)
comp2	character	10	(code name of) actual weapon type in component 2 (if any)
comp3	character	10	(code name of) actual weapon type in component 3 (if any)
comp4	character	10	(code name of) actual weapon type in component 4 (if any)
comp5	character	10	(code name of) actual weapon type in component 5 (if any)
comp6	character	10	(code name of) actual weapon type in component 6 (if any)
val	float	4	data value

Table: RESGENUS ("resources and genuses")
Row width: 20
Description: names of resource categories and associated NAVMOD resource or ordnance variables
Columns and their meanings:

Column Name	Type	Length	Meaning
imeth	integer	1	code number of aggregation method
resnam	character	6	name of main NAVMOD resource variable (for which a notional value is to be computed)
genusnam	character	6	code name of resource category
adlvar	character	6	name of additional NAVMOD variable that may be relevant—the specific use of such variable depends on the aggregation method
imuni	integer	1	further code number for aggregation method

Table: RVARINFO ("real variable information")
Row width: 47
Description: Information on the input variables used by the aggregator (including NAVMOD inputs plus some additional inputs).
Columns and their meanings:

Column Name	Type	Length	Meaning
vnamer	character	6	name of variable
vcoder	integer	1	code number of variable (see documentation for explanation of variable codes and their meanings)
nodr	integer	1	number of dimensions of variable as it is input to the aggregator (0=scalar, 1=one-dimensional array, etc.). Encompasses NAVMOD indexing plus any added indices.
nodn	integer	1	number of dimensions of variable as it is used by NAVMOD
nfn	integer	1	2 for real variables, 1 for integer variables
indx1	character	6	name of index for first dimension (if any)
indx2	character	6	name of index for second dimension (if any)
indx3	character	6	name of index for third dimension (if any)
indx4	character	6	name of index for fourth dimension (if any)
indx5	character	6	name of index for fifth dimension (if any)
indx6	character	6	name of index for sixth dimension (if any)
fcati	integer	1	(not currently used)

Table: SPLITINDG ("split index guide")
Row width: 12
Description: "adaptive" indices for which the resource category (and associated set of actual weapon types) to be used varies depending upon the value of some other ("base") index
Columns and their meanings:

Column Name	Type	Length	Meaning
indxsplit	character	6	name of adaptive index
indxbase	character	6	name of associated "base" index

Table: SPLITINDX ("split indices")
Row width: 28
Description: guide to resource categories for the indices listed in table splitindg, above
Columns and their meanings:

Column Name	Type	Length	Meaning
indxbase	character	6	name of "base" index
indxsplit	character	6	name of adaptive index
bstep	integer	2	step of base index (1, 2,...). Number of steps should equal value in column nsteps of table indxbinfo for the base index.
genusstep	character	6	code name of resource category associated with above step
gvalrl	integer	4	(not currently used)
cumgvalrl	integer	4	(not currently used)

Table: VARDEFS ("variable definitions")
Row width: 77
Description: Definitions of input variables to NAVMOD (does not include variables used by the aggregator but not by NAVMOD). Definitions are identical to those appearing in Appendix F of IDA Paper P-2006. Not used by aggregator itself, but is used by some auxiliary programs. Also used by interaction selector preprocessor.
Columns and their meanings:

Column Name	Type	Length	Meaning
vname	character	6	name of input variable
lineno	integer	1	number of line of definition (1, 2, 3,...)
defline	character	70	text of this line of the definition (possibly continued from previous line)

APPENDIX C

INDEX OF VARIABLES

APPENDIX C

INDEX OF VARIABLES

This appendix lists each input to NAVMOD and the variables that are input to the aggregator but not to NAVMOD. The majority of the variables are simply marked "NAVMOD effectiveness input"; the procedures for handling these have been discused throughout the paper. For non-effectiveness variables and effectiveness variables that need special treatment, this appendix gives references to the text sections that discuss these variables.

AAAEDA	NAVMOD effectiveness input
AAAEDC	NAVMOD effectiveness input
AAAEDE	NAVMOD effectiveness input
AAAEED	NAVMOD effectiveness input
AACA	NAVMOD effectiveness input
AAPAJO	NAVMOD effectiveness input
AAPDDA	NAVMOD effectiveness input
AAPDDC	NAVMOD effectiveness input
AAPDDE	NAVMOD effectiveness input
AAPDED	NAVMOD effectiveness input
AAPKAD	NAVMOD effectiveness input
AAPKDA	NAVMOD effectiveness input
AAPKDC	NAVMOD effectiveness input
AAPKDE	NAVMOD effectiveness input
AAPKED	NAVMOD effectiveness input
AASRAA	NAVMOD effectiveness input
AASRED	NAVMOD effectiveness input
AASRFA	NAVMOD effectiveness input
AASRFE	NAVMOD effectiveness input
AASRID	NAVMOD effectiveness input
ABACAT	NAVMOD effectiveness input
ABALSM	NAVMOD effectiveness input
ABAMAX	NAVMOD effectiveness input
ABAMIN	NAVMOD effectiveness input
ABANM	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d. Also see discussion in Chapter III, Section E.3.d.
ABAPOA	NAVMOD effectiveness input
ABAVLS	NAVMOD effectiveness input
ABEMAX	NAVMOD effectiveness input
ABEMIN	NAVMOD effectiveness input
ABESGS	NAVMOD effectiveness input
ABFASS	NAVMOD effectiveness input
ABFAUS	NAVMOD effectiveness input
ABFSCN	NAVMOD effectiveness input
ABFSM	NAVMOD effectiveness input
ABFVS	NAVMOD effectiveness input
ABMNAO	NAVMOD effectiveness input
ABMNAS	NAVMOD effectiveness input
ABOBAA	NAVMOD effectiveness input
ABOBAG	NAVMOD effectiveness input
ABOBE	NAVMOD effectiveness input
ABORD	NAVMOD effectiveness input
ABPDA	NAVMOD effectiveness input
ABPDS	NAVMOD effectiveness input
ABPKA	NAVMOD effectiveness input
ABPKS	NAVMOD effectiveness input
ABPSA	NAVMOD effectiveness input
ABRSAM	NAVMOD input resource variable—see Chapter II, Section C.3 d.
ABRSMN	NAVMOD effectiveness input
ABRUNA	NAVMOD effectiveness input
ABRUNC	NAVMOD effectiveness input
ABSMSC	NAVMOD effectiveness input
ABTSC	NAVMOD effectiveness input
ABVGSS	NAVMOD effectiveness input
ABWJBA	NAVMOD effectiveness input
ACAVSC	NAVMOD effectiveness input
ACCLCT	NAVMOD effectiveness input. Detail index might be appropriate, see Appendix A, Table A-8; also, Chapter III, Section B.
ACCLSL	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
ACCLTL	NAVMOD effectiveness input
ACCMAT	NAVMOD effectiveness input
ACCSOW	NAVMOD effectiveness input
ACFSCN	NAVMOD effectiveness input
ACPDSC	NAVMOD effectiveness input
ACPKCS	NAVMOD effectiveness input

ACPKSC	NAVMOD effectiveness input
ACTSSC	NAVMOD effectiveness input
ACVRAC	NAVMOD effectiveness input
ACVRSM	NAVMOD effectiveness input
ACWACM	NAVMOD effectiveness input
ACWJBA	NAVMOD effectiveness input
ACXACM	NAVMOD effectiveness input
ADWBSB	NAVMOD effectiveness input
ADWBSS	NAVMOD effectiveness input
AESCAB	NAVMOD input resource variable—see Chapter II, Section C.3.d.
AEWACI	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.3.b.
AEWD	NAVMOD effectiveness input
AINTCP	NAVMOD input resource variable—see Chapter II, Section C.3.d
ARSBA	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.3.c(1).
ASUBAR	NAVMOD effectiveness input
ASUBER	NAVMOD effectiveness input
ASWACI	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.3.b.
ASWF	NAVMOD effectiveness input
ASWSAM	NAVMOD effectiveness input
ASWSLM	NAVMOD effectiveness input
ATABT	NAVMOD input resource variable—see Chapter II, Section C.3.d.
ATTCKI	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.3.b.
AVAILE	NAVMOD effectiveness input
AVAILT	NAVMOD effectiveness input
AVALED	NAVMOD effectiveness input
AVFAIM	NAVMOD effectiveness input
AWHAB	NAVMOD effectiveness input
BACDWB	NAVMOD effectiveness input
BACPCD	NAVMOD effectiveness input
BACPCK	NAVMOD effectiveness input
BARDPH	NAVMOD effectiveness input
BAREAQ	NAVMOD effectiveness input
BARELQ	NAVMOD effectiveness input
BARLQ	NAVMOD effectiveness input
BARLTH	NAVMOD effectiveness input
BECDW	NAVMOD effectiveness input
BEDW	NAVMOD effectiveness input
BESS	NAVMOD input resource variable—see Chapter II, Section C.3.d.
BLAEDA	NAVMOD effectiveness input
BLCPCD	NAVMOD effectiveness input
BLCPCK	NAVMOD effectiveness input
BMTMIN	NAVMOD effectiveness input
BSIBAR	NAVMOD input resource variable—see Chapter II, Section C.3.d.
BSSNDS	NAVMOD input resource variable—see Chapter II, Section C.3.d
BUCAP	NAVMOD effectiveness input
CADWB0	NAVMOD effectiveness input
CAPMLQ	NAVMOD effectiveness input
CAPMQ	NAVMOD effectiveness input
CAPMR	NAVMOD effectiveness input
CAPMV	NAVMOD effectiveness input
CAPSTQ	NAVMOD effectiveness input
COBASA	NAVMOD effectiveness input
COBASL	NAVMOD effectiveness input
COBASR	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
COBCAP	NAVMOD effectiveness input
COBDLI	NAVMOD effectiveness input
CPBPD	NAVMOD effectiveness input
CPBPK	NAVMOD effectiveness input
CPBSCD	NAVMOD effectiveness input
CPBSCK	NAVMOD effectiveness input. Adaptive index might be appropriate; see Chapter II, Section E.3.
CPRPD	NAVMOD effectiveness input
CPRPK	NAVMOD effectiveness input
CPRSCD	NAVMOD effectiveness input

CPRSCK	NAVMOD effectiveness input. Adaptive index might be appropriate; see Chapter II, Section E.3.
CPSMDR	NAVMOD effectiveness input
CPSMES	NAVMOD effectiveness input
CPSMET	NAVMOD effectiveness input
CPSMPD	NAVMOD effectiveness input
CPSMPK	NAVMOD effectiveness input
CPSMTS	NAVMOD effectiveness input
CSCDW0	NAVMOD effectiveness input
CUBASA	NAVMOD effectiveness input
CUBASL	NAVMOD effectiveness input
CUBASR	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
CVREPM	NAVMOD effectiveness input
CVREPP	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.2.a.
CVRPM0	NAVMOD effectiveness input
CXOAAD	NAVMOD effectiveness input
CXOASA	NAVMOD effectiveness input
CXOASF	NAVMOD effectiveness input
CXOASL	NAVMOD effectiveness input
CXOASR	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
CXOCAP	NAVMOD effectiveness input
CXODLI	NAVMOD effectiveness input
CXOSDG	NAVMOD effectiveness input
CXOSSD	NAVMOD effectiveness input
CXUASA	NAVMOD effectiveness input
CXUASL	NAVMOD effectiveness input
CXUASR	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
D1T	NAVMOD effectiveness input. Special restrictions apply; see Chapter IV, Section B.2.d.
D2T	NAVMOD effectiveness input. Special restrictions apply; see Chapter IV, Section B.2.d.
D3T	NAVMOD effectiveness input. Special restrictions apply; see Chapter IV, Section B.2.d.
DDACDT	NAVMOD effectiveness input
DDFKGD	NAVMOD effectiveness input
DDFRAB	NAVMOD effectiveness input. Special restrictions apply; see Chapter IV, Section B.2.d.
DDPRKB	NAVMOD effectiveness input
DDRKAA	NAVMOD effectiveness input
DDRKBA	NAVMOD effectiveness input
DDRSA	NAVMOD effectiveness input
DDSPA	NAVMOD effectiveness input
DINTOP	NAVMOD effectiveness input
DLIA	NAVMOD effectiveness input
DLIALR	NAVMOD effectiveness input
DLIMR	NAVMOD effectiveness input
DLIMV	NAVMOD effectiveness input
DMCAPS	NAVMOD effectiveness input
DURGS	NAVMOD effectiveness input
DURGS0	NAVMOD effectiveness input
EAAFRF	NAVMOD effectiveness input
EAAKAD	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.3.a. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
EAAKDA	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
EAAKED	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
EAAMPA	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.3.a.
EAAMPD	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
EAAMPE	NAVMOD effectiveness input
EAAWFE	NAVMOD effectiveness input

EAAWFF	NAVMOD effectiveness input. Detail index might be appropriate, see Appendix A, Table A-8; also, Chapter III, Section B
EAFSRQ	NAVMOD effectiveness input
EAOBAA	NAVMOD effectiveness input See the discussion of this input in Chapter IV, Section B.3.a.
EAOBAF	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.3.a.
EAOBEA	NAVMOD effectiveness input
EAORDA	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
EAVRA	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
EAVSRQ	NAVMOD effectiveness input
ELAEAD	NAVMOD effectiveness input
ELAESD	NAVMOD effectiveness input
ELAFDM	NAVMOD effectiveness input
ELAFVL	NAVMOD effectiveness input
ELAKAD	NAVMOD effectiveness input
ELAKSD	NAVMOD effectiveness input
ELAOAD	NAVMOD effectiveness input
ELAOSD	NAVMOD effectiveness input
ELAPK	NAVMOD effectiveness input
ELAPMU	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.3.a.
ELASPS	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.3.a.
ELATW	NAVMOD effectiveness input
ELBEA	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.3.a.
ELBEE	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.1.a.
ELFAAV	NAVMOD effectiveness input
ELFASW	NAVMOD effectiveness input
ELFBAK	NAVMOD effectiveness input
ELFBL0	NAVMOD effectiveness input. Special restrictions apply, see Chapter IV, Section B.2.d.
ELFBLA	NAVMOD effectiveness input
ELFNVS	NAVMOD effectiveness input
ELFRSV	NAVMOD effectiveness input
ELFSLN	NAVMOD effectiveness input
ELFVS	NAVMOD effectiveness input
ELFWPL	NAVMOD effectiveness input
ELLEAD	NAVMOD effectiveness input
ELLFAV	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.3.a.
ELLFDM	NAVMOD effectiveness input
ELLKAD	NAVMOD effectiveness input
ELLOAD	NAVMOD effectiveness input
ELORAF	NAVMOD effectiveness input
ELPPSC	NAVMOD effectiveness input
ELWASG	NAVMOD effectiveness input
ENACDS	NAVMOD effectiveness input
ENACDT	NAVMOD effectiveness input Adaptive index might be appropriate, see Chapter II, Section E.3
ESCMIN	NAVMOD effectiveness input
ESLF	NAVMOD effectiveness input
ESRC	NAVMOD effectiveness input
FAACA	NAVMOD effectiveness input
FABRSM	Allocation fraction variable. Input to aggregator but not to NAVMOD. See Chapter II, Section C.3.c. Also see discussion in Chapter III, Section E.3.d.
FACOB	NAVMOD effectiveness input
FACOBC	NAVMOD effectiveness input
FASWLB	NAVMOD effectiveness input
FATABT	Allocation fraction variable. Input to aggregator but not to NAVMOD. See Chapter II, Section C.3.c. Also see Chapter IV, Section B.2.d.
FBESS	Allocation fraction variable. Input to aggregator but not to NAVMOD. See Chapter II, Section C.3.c.

FBSESO	NAVMOD effectiveness input
FFACA	NAVMOD effectiveness input
FFACE	NAVMOD effectiveness input
FGHTRI	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.3.b.
FHSK	NAVMOD effectiveness input
FHSKC	NAVMOD effectiveness input
FM3	NAVMOD effectiveness input
FPLBLB	Allocation fraction variable. Input to aggregator but not to NAVMOD. See Chapter II, Section C.3.c. Also see the discussion of this input in Chapter IV, Section B.1.a.
FPPL1	NAVMOD effectiveness input
FPPL2	NAVMOD effectiveness input
FPPRSM	Allocation fraction variable. Input to aggregator but not to NAVMOD. See Chapter II, Section C.3.c. Also see discussion in Chapter III, Section E.3.d.
FRMSET	NAVMOD effectiveness input
FRRSPA	NAVMOD effectiveness input
FRSFNR	Allocation fraction variable. Input to aggregator but not to NAVMOD See Chapter II, Section C.3.c and Chapter IV, Section D.1.b(2).
FRSSG	Allocation fraction variable. Input to aggregator but not to NAVMOD. See Chapter II, Section C.3.c.
FTBBA	NAVMOD effectiveness input
FTRBA	NAVMOD effectiveness input
FTRMRM	NAVMOD effectiveness input
FURGA	NAVMOD effectiveness input
FURGA0	NAVMOD effectiveness input
FURGT0	Declared as a NAVMOD input but not actually used in the NAVMOD code; not processed by aggregator.
FURGTP	Declared as a NAVMOD input but not actually used in the NAVMOD code; not processed by aggregator.
GNFRUN	NAVMOD effectiveness input
GNORAA	NAVMOD effectiveness input
GNORAF	NAVMOD effectiveness input
GNORE	NAVMOD effectiveness input
HRMBSS	NAVMOD effectiveness input
HRTBSS	NAVMOD effectiveness input
IAADA	Declared as a NAVMOD input but not actually used in the NAVMOD code; not processed by aggregator.
IAAED	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IABAD1	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IABAEQ	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IABAF	Declared as a NAVMOD input but not actually used in the NAVMOD code; not processed by aggregator.
IABAW	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IAEWDF	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IATKRT	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IATRIA	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
ICTFXP	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
ICTL	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IDDAS	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IEAAF	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IELAAM	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IELASG	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IELRGI	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IELXP	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IPIADA	Declared as a NAVMOD input but not actually used in the NAVMOD code; not processed by aggregator.
IPIAED	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IPICDA	Declared as a NAVMOD input but not actually used in the NAVMOD code; not processed by aggregator.
IPLADA	Declared as a NAVMOD input but not actually used in the NAVMOD code; not processed by aggregator.
IPLAED	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IPPAF	Declared as a NAVMOD input but not actually used in the NAVMOD code; not processed by aggregator.
IPPAW	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.

IPPED1	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IPRSRA	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a. Also see the discussion of this input in Chapter IV, Section B.3.c(1).
IRKO	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IRNCPD	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IRPBB	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IRPNTE	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IRPRB	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IRPRG	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IRSBKP	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IRSCKA	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IRSCKP	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IRSOD1	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IRSSF	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3 a.
IRSUBA	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IRSXP	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
ISAAF	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
ISAAM	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
ISBKPF	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
ISBKPS	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
ISBXP	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
ISCKPF	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
ISCKPS	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
ISLOC	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
ISSBR	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
ISSCTF	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
ISSEAM	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
ISSLAM	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
ISSRB	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
ISSXP	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
ITFFKM	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IVTBKP	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IVTCKA	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IVTCKL	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IVTCKP	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
IVTXP	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
LGSGMP	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
LGTHMP	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
LSAGMP	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
LTFMP	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
LURGSN	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
MAXTP	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
MIMP	NAVMOD input limit variable—see Chapter II, Section C.3.b.
MIMPSG	NAVMOD input limit variable, but is not input to aggregator. See Chapter II, Section C.3.b, Chapter III, Section C, and Chapter IV, Section B.2.f.
NABSAM	NAVMOD input limit variable—see Chapter II, Section C.3.b.
NCABE	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
NCABL	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
NCPPE	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3 a.
NCPPL	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a.
NKBDPL	NAVMOD input limit variable—see Chapter II, Section C.3.b. Also see the discussion of this input in Chapter IV, Section B.1.a.
NKBES	NAVMOD input limit variable—see Chapter II, Section C.3.b. Also see Chapter IV, Section B.2.f.
NKRB	NAVMOD input limit variable—see Chapter II, Section C.3.b.
NKRS	NAVMOD input limit variable, but is not input to aggregator. See Chapter II, Section C.3.b, Chapter III, Section C, and Chapter IV, Section B.2.f.
NKRSN	Special limit variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.b, Chapter III, Section C, Chapter IV, Section B.2.f, and Chapter IV, Section D.1.b(2).
NLOC	NAVMOD input limit variable—see Chapter II, Section C.3.b. Variable appears in computer code; see Chapter III, Section E.3. Also see Chapter IV, Section D.1.b(3).
NMPSG	Special limit variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.b and Chapter III, Section C

NPPFFX	NAVMOD input limit variable—see Chapter II, Section C.3.b. Special restrictions apply; see Chapter IV, Section B.2.d. Also see Chapter IV, Section B.2.g.
NPPSAM	NAVMOD input limit variable—see Chapter II, Section C.3.b
NSAG	NAVMOD input limit variable—see Chapter II, Section C.3.b. Also see Chapter IV, Sections B.2.f and B.3.c(5).
NSAGAS	NAVMOD integer-valued effectiveness input—see Chapter II, Section C.3.a
NTABA	NAVMOD input limit variable—see Chapter II, Section C.3.b. Also see Chapter IV, Section B.2.g.
NTABD	NAVMOD input limit variable—see Chapter II, Section C.3.b. Also see Chapter IV, Section B.2.g.
NTABH	NAVMOD input limit variable—see Chapter II, Section C.3.b. Also see Chapter IV, Section B.2.g.
NTABS	NAVMOD input limit variable—see Chapter II, Section C.3.b. Also see Chapter IV, Section B.2.g
NTABV	NAVMOD input limit variable—see Chapter II, Section C.3.b. Also see Chapter IV, Section B.2.g
NTC	NAVMOD input limit variable—see Chapter II, Section C.3.b. Also see Chapter IV, Section B.2.f
OIBAAA	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIBAAF	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIBAAG	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIBAAS	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d
OIBBAR	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIBFCM	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIBFDA	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIBFDS	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIBFFF	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIBFTP	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIBSCM	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIBSTP	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIBUA	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIBUR	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIRBAR	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIRFAA	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIRFAS	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIRFDA	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIRFDL	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIRFDS	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIRFFL	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIRFFR	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIRFTP	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIRSCM	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OIRSTP	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OVBAAA	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21.
OVBAAF	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21.
OVBAAG	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21.
OVBAAS	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21
OVBBAR	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d
OVBFCM	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21
OVBFDA	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21
OVBFDS	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21
OVBFFF	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21
OVBFTP	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21
OVBLAA	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OVBLAF	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d
OVBLAS	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.

OVBSCM	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21.
OVBSTP	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21.
OVBTF	NAVMOD reserve ordnance input, but is not input to aggregator. See Chapter II, Section C.3.d; also see the discussion in Chapter III, Sections E.2 and E.3.g.
OVBUL	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OVRBAR	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OVRFAA	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21.
OVRFAS	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21
OVRFDA	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21.
OVRFDL	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21.
OVRFDS	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21.
OVRFFL	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21.
OVRFFR	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21.
OVRFTP	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21.
OVRLAA	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OVRLAF	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d.
OVRRG	NAVMOD reserve ordnance input, but is not input to aggregator. See Chapter II, Section C.3.d; also see the discussion in Chapter III, Sections E.2 and E.3.h.
OVRSCM	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21.
OVRSTP	Special ordnance variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21.
PABORT	NAVMOD effectiveness input
PAFCNF	NAVMOD effectiveness input
PARK	NAVMOD effectiveness input
PASS	NAVMOD effectiveness input
PBCMF	NAVMOD effectiveness input
PBDRN	NAVMOD effectiveness input
PBDRS	NAVMOD effectiveness input
PBKRN	NAVMOD effectiveness input
PBKRNC	NAVMOD effectiveness input
PBKRS	NAVMOD effectiveness input
PBKRSC	NAVMOD effectiveness input
PDINV	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.2.g.
PFFCNF	NAVMOD effectiveness input
PIAEDA	NAVMOD effectiveness input
PIAEDC	NAVMOD effectiveness input
PIAEDE	NAVMOD effectiveness input
PIAEED	NAVMOD effectiveness input
PICAL	NAVMOD effectiveness input
PIESPF	NAVMOD effectiveness input
PIESPI	NAVMOD effectiveness input
PIFFA	NAVMOD effectiveness input
PIFRIF	NAVMOD effectiveness input
PIORD	NAVMOD effectiveness input
PIPAJO	NAVMOD effectiveness input
PIPDDA	NAVMOD effectiveness input
PIPDDC	NAVMOD effectiveness input
PIPDDE	NAVMOD effectiveness input
PIPDED	NAVMOD effectiveness input
PIPKAD	NAVMOD effectiveness input
PIPKDA	NAVMOD effectiveness input
PIPKDC	NAVMOD effectiveness input
PIPKDE	NAVMOD effectiveness input

PIPKED	NAVMOD effectiveness input
PISORR	NAVMOD effectiveness input
PIWACM	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.3.c(4).
PKAAD	NAVMOD effectiveness input
PKASW	NAVMOD effectiveness input
PKAT1	NAVMOD effectiveness input. Adaptive index might be appropriate; see Chapter II, Section E.3.
PKBF1	NAVMOD effectiveness input
PKDF1	NAVMOD effectiveness input
PKFASM	NAVMOD effectiveness input
PKHBSS	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
PKIIN	NAVMOD effectiveness input
PKPL1	NAVMOD effectiveness input
PKPL2	NAVMOD effectiveness input
PKPLDT	NAVMOD effectiveness input. Adaptive index might be appropriate; see Chapter II, Section E.3.
PKSAAD	Declared as a NAVMOD input but not actually used in the NAVMOD code; not processed by aggregator.
PLAEDA	NAVMOD effectiveness input
PLAEDE	NAVMOD effectiveness input
PLAEED	NAVMOD effectiveness input
PLBLBD	NAVMOD input resource variable—see Chapter II, Section C.3.a.
PLCA	NAVMOD effectiveness input
PLFDLL	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.1.a.
PLFEBO	NAVMOD effectiveness input
PLOBD	NAVMOD effectiveness input
PLPAJO	NAVMOD effectiveness input
PLPDDA	NAVMOD effectiveness input
PLPDDE	NAVMOD effectiveness input
PLPDED	NAVMOD effectiveness input
PLPKAD	NAVMOD effectiveness input
PLPKDA	NAVMOD effectiveness input
PLPKDE	NAVMOD effectiveness input
PLPKED	NAVMOD effectiveness input
PPAEGS	NAVMOD effectiveness input
PPALSM	NAVMOD effectiveness input
PPANMS	NAVMOD input ordnance stock variable—see Chapter II, Section C.3.d. Also see discussion in Chapter III, Section E.3 d.
PPASTA	NAVMOD effectiveness input
PPAVLS	NAVMOD effectiveness input
PPAVSC	NAVMOD effectiveness input
PPAVSS	NAVMOD effectiveness input
PPCLCT	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
PPCLSL	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
PPCLTL	NAVMOD effectiveness input
PPCVFX	NAVMOD effectiveness input. Special restrictions apply; see Chapter IV, Section B.2.c. Also see Chapter IV, Section B.2.g.
PPFASM	NAVMOD effectiveness input
PPFASS	NAVMOD effectiveness input
PPFCSS	NAVMOD effectiveness input
PPFRUN	NAVMOD effectiveness input
PPFSVS	NAVMOD effectiveness input
PPIVMV	NAVMOD effectiveness input
PPMNAS	NAVMOD effectiveness input
PPOBAA	NAVMOD effectiveness input
PPOBAG	NAVMOD effectiveness input
PPOBE	NAVMOD effectiveness input
PPPDAS	NAVMOD effectiveness input
PPPDSA	NAVMOD effectiveness input
PPPDSC	NAVMOD effectiveness input
PPPKAS	NAVMOD effectiveness input
PPPKCS	NAVMOD effectiveness input
PPPKSA	NAVMOD effectiveness input

PPPKSC	NAVMOD effectiveness input
PPPSAS	NAVMOD effectiveness input
PPRSAM	NAVMOD input resource variable—see Chapter II, Section C.3.d.
PPSMAX	NAVMOD effectiveness input
PPSMIN	NAVMOD effectiveness input
PPSORR	NAVMOD effectiveness input
PPSSFX	NAVMOD effectiveness input
PPTATA	NAVMOD effectiveness input
PPTGTA	NAVMOD effectiveness input
PPTSCS	NAVMOD effectiveness input
PPTSSC	NAVMOD effectiveness input
PPVRSM	NAVMOD effectiveness input
PPWFTC	NAVMOD effectiveness input
PRSM	NAVMOD effectiveness input
PRWLNQ	NAVMOD effectiveness input
PSLCMN	NAVMOD effectiveness input
QBAAA	NAVMOD effectiveness input—ordnance capacity
QBAAF	NAVMOD effectiveness input—ordnance capacity
QBAAG	NAVMOD effectiveness input—ordnance capacity
QBAAS	NAVMOD effectiveness input—ordnance capacity
QBBAR	NAVMOD effectiveness input—ordnance capacity
QBFCM	NAVMOD effectiveness input—ordnance capacity
QBFDA	NAVMOD effectiveness input—ordnance capacity
QBFDS	NAVMOD effectiveness input—ordnance capacity
QBFFF	NAVMOD effectiveness input—ordnance capacity
QBFTP	NAVMOD effectiveness input—ordnance capacity
QBSCM	NAVMOD effectiveness input—ordnance capacity
QBSTP	NAVMOD effectiveness input—ordnance capacity
QRBAR	NAVMOD effectiveness input—ordnance capacity
QRFAA	NAVMOD effectiveness input—ordnance capacity
QRFAS	NAVMOD effectiveness input—ordnance capacity
QRFDA	NAVMOD effectiveness input—ordnance capacity
QRFDL	NAVMOD effectiveness input—ordnance capacity
QRFDS	NAVMOD effectiveness input—ordnance capacity
QRFFL	NAVMOD effectiveness input—ordnance capacity
QRFFR	NAVMOD effectiveness input—ordnance capacity
QRFTP	NAVMOD effectiveness input—ordnance capacity
QRSCM	NAVMOD effectiveness input—ordnance capacity
QRSTP	NAVMOD effectiveness input—ordnance capacity
QWBBAR	NAVMOD effectiveness input
QWBTF	NAVMOD effectiveness input
QWRBAR	NAVMOD effectiveness input
QWRRG	NAVMOD effectiveness input
QXBBAR	NAVMOD effectiveness input
QXBURG	NAVMOD effectiveness input
QXRBAR	NAVMOD effectiveness input
QXRRSP	NAVMOD effectiveness input
RACAB	NAVMOD effectiveness input
RACCDW	NAVMOD effectiveness input. Detail index might be appropriate, see Appendix A, Table A-8; also, Chapter III, Section B
RACPCD	NAVMOD effectiveness input. Detail index might be appropriate, see Appendix A, Table A-8, also, Chapter III, Section B
RACPCK	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8, also, Chapter III, Section B.
RAPRS	NAVMOD effectiveness input. See the discussion in Chapter IV, Section B.3.c(1). Also detail index might be appropriate, see Appendix A, Table A-8; also, Chapter III, Section B
RARBAB	NAVMOD effectiveness input
RARCAB	NAVMOD effectiveness input
RECDW	NAVMOD effectiveness input
REDW	NAVMOD effectiveness input
RS	NAVMOD resource input, but is not input to aggregator. See Chapter II, Section C.3.d; also see discussion in Chapter III, Section E.2.
RSAPRP	NAVMOD effectiveness input
RSBSF	NAVMOD effectiveness input
RSBTP	Special resource variable, input to aggregator but not to NAVMOD. See Chapter II, Section C.3.d, especially Table II-21.
RSFBSA	NAVMOD effectiveness input

RSFNRP	Special resource variable, input to aggregator but not to NAVMOD See Chapter II, Section C.3.d, especially Table II-21.
RSFRSP	Special resource variable, input to aggregator but not to NAVMOD See Chapter II, Section C.3.d, especially Table II-21.
RSIBAR	NAVMOD input resource variable—see Chapter II, Section C.3.d
RSMFB	NAVMOD effectiveness input
RSOBBS	NAVMOD effectiveness input
RSOBX	NAVMOD effectiveness input
RSORA	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
RSORF	NAVMOD effectiveness input
RSORS	NAVMOD effectiveness input
RSSG	NAVMOD input resource variable—see Chapter II, Section C.3.d
RSSPBS	Declared as a NAVMOD input but not actually used in the NAVMOD code; not processed by aggregator.
RSSPRP	NAVMOD effectiveness input
RSWBB	NAVMOD effectiveness input
RSWRS	NAVMOD effectiveness input
RUNCLO	NAVMOD effectiveness input
RUNDAB	NAVMOD effectiveness input
RUNDAM	NAVMOD effectiveness input
RUNDPG	NAVMOD effectiveness input
RUNRCP	NAVMOD input resource variable—see Chapter II, Section C.3.d.
RUNRDG	NAVMOD effectiveness input
RUNREP	NAVMOD effectiveness input
RURGS	NAVMOD input resource variable—see Chapter II, Section C.3.d. Also see discussion in Chapter III, Section E.3.g.
SAAFRF	NAVMOD effectiveness input
SAAKAD	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
SAAKDA	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
SAAKED	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
SAAMPA	NAVMOD effectiveness input
SAAMPD	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
SAAMPE	NAVMOD effectiveness input
SAAWFE	NAVMOD effectiveness input
SAAWFF	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
SAFACM	NAVMOD effectiveness input
SAFSRQ	NAVMOD effectiveness input
SAOBAA	NAVMOD effectiveness input
SAOBAF	NAVMOD effectiveness input
SAOBEA	NAVMOD effectiveness input
SAORDA	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B
SAPBSA	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.3.c(5)
SASORR	NAVMOD effectiveness input
SAVRA	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B
SAVSRC	NAVMOD effectiveness input
SBBSFF	NAVMOD effectiveness input
SBBSFS	NAVMOD effectiveness input
SBCDWF	NAVMOD effectiveness input
SBCDWS	NAVMOD effectiveness input
SBCWF0	NAVMOD effectiveness input
SBCWS0	NAVMOD effectiveness input
SBEDWF	NAVMOD effectiveness input
SBEDWS	NAVMOD effectiveness input
SBFBSA	NAVMOD effectiveness input
SBFRSA	NAVMOD effectiveness input
SBMFD	NAVMOD effectiveness input
SBOBD	NAVMOD effectiveness input
SBOBXF	NAVMOD effectiveness input

SBOBXS NAVMOD effectiveness input
SBORCM NAVMOD effectiveness input
SBORF NAVMOD effectiveness input
SBORS NAVMOD effectiveness input
SBORTC NAVMOD effectiveness input
SBORTP NAVMOD effectiveness input
SBPBDF NAVMOD effectiveness input
SBPBDS NAVMOD effectiveness input
SBPBKF NAVMOD effectiveness input
SBPBKS NAVMOD effectiveness input
SBPFDB NAVMOD effectiveness input
SBPFKB NAVMOD effectiveness input
SBPSDB NAVMOD effectiveness input
SBPSKB NAVMOD effectiveness input
SBSCR NAVMOD effectiveness input
SBSPRF NAVMOD effectiveness input
SBSPRS NAVMOD effectiveness input
SBTHSC NAVMOD effectiveness input
SBWBSF NAVMOD effectiveness input
SBWBSS NAVMOD effectiveness input
SBWRFS NAVMOD effectiveness input
SBWRSS NAVMOD effectiveness input
SDSBSG NAVMOD effectiveness input
SDSBSS NAVMOD effectiveness input
SGSRS NAVMOD effectiveness input
SHEL NAVMOD input resource variable—see Chapter II, Section C.3.d.
SHELE NAVMOD effectiveness input
SLCCS NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.3.c(3).
SLCTT NAVMOD effectiveness input
SPA NAVMOD effectiveness input
SPAEW NAVMOD effectiveness input
SPASW NAVMOD effectiveness input
SPF NAVMOD effectiveness input
SSAEAD NAVMOD effectiveness input
SSAESD NAVMOD effectiveness input
SSAFDM NAVMOD effectiveness input
SSAFVL NAVMOD effectiveness input
SSAKAD NAVMOD effectiveness input
SSAKSD NAVMOD effectiveness input
SSAOAD NAVMOD effectiveness input
SSAOSD NAVMOD effectiveness input
SSAPK NAVMOD effectiveness input
SSAPMU NAVMOD effectiveness input
SSASPS NAVMOD effectiveness input
SSATW NAVMOD effectiveness input
SSBREA NAVMOD effectiveness input
SSBRES NAVMOD effectiveness input
SSBRFD NAVMOD effectiveness input
SSBRFV NAVMOD effectiveness input
SSBRKA NAVMOD effectiveness input
SSBRKS NAVMOD effectiveness input
SSBROA NAVMOD effectiveness input
SSBROS NAVMOD effectiveness input
SSBRPK NAVMOD effectiveness input
SSBRPU NAVMOD effectiveness input
SSBRTW NAVMOD effectiveness input
SSBSEO NAVMOD effectiveness input
SSCFA NAVMOD effectiveness input
SSEEAD NAVMOD effectiveness input
SSEESD NAVMOD effectiveness input
SSEFDM NAVMOD effectiveness input
SSEFVL NAVMOD effectiveness input
SSEKAD NAVMOD effectiveness input
SSEKSD NAVMOD effectiveness input
SSEOAD NAVMOD effectiveness input
SSEOSD NAVMOD effectiveness input

SSEPD	NAVMOD effectiveness input
SSEPK	NAVMOD effectiveness input
SSEPMU	NAVMOD effectiveness input
SSESPS	NAVMOD effectiveness input
SSETW	NAVMOD effectiveness input
SSFBSF	NAVMOD effectiveness input
SSFDCT	NAVMOD effectiveness input
SSFEAK	NAVMOD effectiveness input. Special restrictions apply; see Chapter IV, Section B.2.d.
SSFEL	NAVMOD effectiveness input
SSFKGD	NAVMOD effectiveness input
SSFNLE	NAVMOD effectiveness input
SSFRSV	NAVMOD effectiveness input
SSFUAK	NAVMOD effectiveness input. Special restrictions apply. see Chapter IV, Section B.2.d.
SSFVLE	NAVMOD effectiveness input
SSLEAD	NAVMOD effectiveness input
SSLFAV	NAVMOD effectiveness input
SSLFDM	NAVMOD effectiveness input
SSLKAD	NAVMOD effectiveness input
SSLOAD	NAVMOD effectiveness input
SSPBDR	NAVMOD effectiveness input
SSPBKR	NAVMOD effectiveness input
SSPRDB	NAVMOD effectiveness input
SSPRKB	NAVMOD effectiveness input
SSRBEA	NAVMOD effectiveness input
SSRBES	NAVMOD effectiveness input
SSRBFD	NAVMOD effectiveness input
SSRBFV	NAVMOD effectiveness input
SSRBKA	NAVMOD effectiveness input
SSRBKS	NAVMOD effectiveness input
SSRBOA	NAVMOD effectiveness input
SSRBOS	NAVMOD effectiveness input
SSRBPK	NAVMOD effectiveness input
SSRBPU	NAVMOD effectiveness input
SSRBTW	NAVMOD effectiveness input
SSSORR	NAVMOD effectiveness input. Special restrictions apply; see Chapter IV, Section B.2.d.
SSSPBS	NAVMOD effectiveness input
SSSPRS	NAVMOD effectiveness input
SSTWBR	NAVMOD effectiveness input
SSTWRB	NAVMOD effectiveness input
SSUEAR	NAVMOD effectiveness input
SSWBFF	NAVMOD effectiveness input
SSWRFF	NAVMOD effectiveness input
STARQ	NAVMOD effectiveness input
STSALV	NAVMOD effectiveness input
SUBSOR	NAVMOD effectiveness input
T1	NAVMOD effectiveness input
T2	NAVMOD effectiveness input
T3	NAVMOD effectiveness input
T4	NAVMOD effectiveness input
TAB10T	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.2.g
TAB12	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.2.g
TAB13T	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.2.g. Also, an adaptive index might be appropriate for this input, see Chapter II, Section E.3
TCAP	NAVMOD effectiveness input
TELFBA	NAVMOD effectiveness input
TELFTV	NAVMOD effectiveness input
TELPD	NAVMOD effectiveness input
TELPK	NAVMOD effectiveness input
TFAAWE	NAVMOD effectiveness input

TFAPTS	NAVMOD effectiveness input
TFASWE	NAVMOD effectiveness input
TFASWS	NAVMOD effectiveness input
TFFADM	NAVMOD effectiveness input
TFFASW	NAVMOD effectiveness input
TFFBSF	NAVMOD effectiveness input
TFFDCT	NAVMOD effectiveness input
TFFELL	NAVMOD effectiveness input
TFFFVM	NAVMOD effectiveness input
TFFFVT	NAVMOD effectiveness input
TFFKGD	NAVMOD effectiveness input
TFMDAS	NAVMOD effectiveness input
TFWFES	NAVMOD effectiveness input
TFWFFT	NAVMOD effectiveness input
THSCAQ	NAVMOD effectiveness input
THSCTC	NAVMOD effectiveness input
TOBBAR	NAVMOD input resource variable—see Chapter II, Section C.3.d.
TORBAR	NAVMOD input resource variable—see Chapter II, Section C.3.d
TPAS	NAVMOD effectiveness input
TPS	NAVMOD effectiveness input
TRSBA0	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.3.c(1).
TRSFPA	NAVMOD effectiveness input
TRSFTV	NAVMOD effectiveness input
TRSPD	NAVMOD effectiveness input
TRSPK	NAVMOD effectiveness input
TVTFPA	NAVMOD effectiveness input
TVTFTV	NAVMOD effectiveness input
TVTPD	NAVMOD effectiveness input
TVTPK	NAVMOD effectiveness input
UAFATF	NAVMOD effectiveness input
UAFTTP	NAVMOD effectiveness input
UAPDSB	NAVMOD effectiveness input
UAPDSR	NAVMOD effectiveness input
UAPDSS	NAVMOD effectiveness input
UAPKRS	NAVMOD effectiveness input. Adaptive index might be appropriate; see Chapter II, Section E.3.
UAPKSB	NAVMOD effectiveness input
UBAEW	NAVMOD effectiveness input
UBAEWL	NAVMOD effectiveness input
UBASW	NAVMOD effectiveness input
UBASWL	NAVMOD effectiveness input
URGEP	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.2.a.
VBT	NAVMOD effectiveness input. Special restrictions apply; see Chapter IV, Section B.2.d.
VCAP	NAVMOD effectiveness input
VI	NAVMOD effectiveness input
VMT	NAVMOD effectiveness input. Special restrictions apply; see Chapter IV, Section B.2.d.
VRAB	NAVMOD effectiveness input
VTAPBL	NAVMOD effectiveness input
VTAPBP	NAVMOD effectiveness input
VTFBSA	NAVMOD effectiveness input
VTFDCT	NAVMOD effectiveness input
VTFKGD	NAVMOD effectiveness input
VTMFB	NAVMOD effectiveness input
VTOBL	NAVMOD effectiveness input
VTOBLX	NAVMOD effectiveness input
VTOBPA	NAVMOD effectiveness input
VTOBPF	NAVMOD effectiveness input
VTOBPS	NAVMOD effectiveness input
VTORBS	NAVMOD effectiveness input
VTRSF	NAVMOD effectiveness input
VTRTWC	NAVMOD effectiveness input

VTSPBP	NAVMOD effectiveness input
VTSPBS	Declared as a NAVMOD input but not actually used in the NAVMOD code; not processed by aggregator.
VTTAB	NAVMOD effectiveness input. See the discussion of this input in Chapter IV, Section B.2.g.
VTUBL	NAVMOD effectiveness input
VTUBLX	NAVMOD effectiveness input
VTWBS	NAVMOD effectiveness input
VTWRB	NAVMOD effectiveness input
WFMBST	NAVMOD effectiveness input
WFPPAS	NAVMOD effectiveness input
WFPPCM	NAVMOD effectiveness input
WFTBSS	NAVMOD effectiveness input
WFTFL	NAVMOD effectiveness input
WRLNDQ	NAVMOD effectiveness input
WRSCB0	NAVMOD effectiveness input
WTFCB0	NAVMOD effectiveness input
WVSIZ	NAVMOD effectiveness input
XAEW	NAVMOD input resource variable—see Chapter II, Section C.3.d.
XAEWLQ	NAVMOD input resource variable—see Chapter II, Section C.3.d.
XASW	NAVMOD input resource variable—see Chapter II, Section C.3.d.
XASWLQ	NAVMOD input resource variable—see Chapter II, Section C.3.d.
XATTCK	NAVMOD input resource variable—see Chapter II, Section C.3.d.
XEFCDI	NAVMOD effectiveness input
XEFCDL	NAVMOD effectiveness input
XEFCKI	NAVMOD effectiveness input
XEFCKL	NAVMOD effectiveness input
XEFCSL	NAVMOD effectiveness input
XFGHTR	NAVMOD input resource variable—see Chapter II, Section C.3.d.
XPLAT	NAVMOD input resource variable—see Chapter II, Section C.3.d.
XURGS	NAVMOD input resource variable—see Chapter II, Section C.3.d.
ZLAMPF	NAVMOD effectiveness input. Detail index might be appropriate; see Appendix A, Table A-8; also, Chapter III, Section B.
ZMAATT	NAVMOD effectiveness input
ZMPATT	NAVMOD effectiveness input
ZMPCAP	NAVMOD effectiveness input
ZMPDLI	NAVMOD effectiveness input
ZMPESC	NAVMOD effectiveness input
ZMPSTG	NAVMOD effectiveness input

DISTRIBUTION

IDA Paper P-2259

AN AGGREGATOR PREPROCESSOR FOR NAVMOD

60 Copies

DEPARTMENT OF DEFENSE

Office Joint Chiefs of Staff
Washington, D.C. 20301-5000

ATTN:	Arthur W. Paarmann, J-8	20
	Director, Information and Resource Management	10

Defense Technical Information Center Cameron Station Alexandria, VA 22315	2
Institute for Defense Analyses 1801 N. Beauregard Street Alexandria, VA 22311	28

ATTN:	General W.Y. Smith	1
	Mr. Philip L. Major	1
	Dr. Robert E. Roberts	1
	Dr. W.J. Schultis	1
	Dr. J.H. Grotte	1
	Dr. L.B. Anderson	1
	Dr. R.J. Atwell	1
	Mr. J.M. Cook	1
	Ms. S. Smith	1
	Dr. P. Gould	1
	Mr. E.P. Kerlin	1
	Mr. D.G. McBryde	1
	Dr. F.A. Miercort	1
	Ms. M.M. Pett	1
	Mr. M.L. Roberson	1
	Dr. L.A. Schmidt	1
	Ms. E.L. Schwartz	5
	Dr. L.D. Simmons	1
	Control & Distribution	6